UNIVERSIDAD AUTONOMA
DE MADRID

Escuela Politécnica Superior
Departamento de Tecnología Electrónica
y de las Comunicaciones

MECANISMOS DE COOPERACIÓN EN ROBOTS COMO AGENTES MÓVILES

TESIS DOCTORAL

Guillermo González de Rivera Peces

Madrid, octubre de 2011

A la memoria de mis padres

Agradecimientos

Llegó la hora. Parecía que no iba a llegar nunca, pero aquí está. Por fin me he decidido a juntar en un tomo el resultado de todo este trabajo que hemos venido haciendo desde hace ya algunos años. Y digo "hemos" en lugar de "he" porque, afortunadamente, en este trabajo no he estado solo, sino acompañado y, por cierto, muy bien acompañado.

Yo he puesto la ilusión de hacer algo que me apasionaba y el resto de compañeros, empezando por mi Director de Tesis, me han apoyado, me han resuelto dudas, me han aportado nuevas ideas, han revisado artículos (principalmente el inglés) y sobre todo, me han animado hasta límites insospechados en algún momento de bajón que he padecido. En definitiva, han estado conmigo siempre que los he necesitado.

Me han recomendado que los agradecimientos no ocuparan más que el resto de texto de la tesis, por lo que no voy a poder agradecerle a Javier Garrido, mi Director de Tesis, todo lo que le debo, todo lo que ha hecho por mí. Desde que llegué a la UAM siempre ha estado ahí, conmigo. Me integró completamente en su mundo y siempre he sentido su apoyo, su ánimo, su cariño y su respeto. No siempre han sido buenos momentos, me he llevado alguna que otra bronca, pero he de reconocer que yo tampoco se lo puse fácil. Desde luego siempre ha sido más que mi director de tesis y espero que lo siga siendo por muchos años más, aunque entendería que después de la guerra que le he dado quisiera pedir la jubilación anticipada, por si le llega otro como yo… Gracias Javier, ambos sabemos que sin tu ánimo y apoyo no hubiera llegado hasta aquí.

Tampoco quiero dejar pasar la ocasión de dar las gracias a la persona que, aún sin acabar la carrera, me dio la primera oportunidad de acercarme al mundo universitario, Eduardo Boemo. Me llamó para formar parte del selecto mundo de los monitores de laboratorio que curraban gratis en la UPM (era otra época). Yo aún no sabía eso, lo de currar gratis, pero desde luego, si hoy tuviera que tomar esa decisión, volvería a ser la misma. Me puso en contacto con la investigación, con la docencia, con la difusión de todo ese conocimiento a las empresas y, en cuanto tuvo ocasión, tiró de mí para venir a la UAM. Gracias Eduardo por la confianza y el cariño que siempre me has demostrado.

También quiero mencionar a las personas que directamente han colaborado en la tesis de alguna manera. En particular, gracias Ricardo por tu inestimable ayuda con Linux, XML-RPC, CGIs, APIS, Apaches, Boas, TCP, IP, RPC, HTTP y tantas y tantas siglas de las que casi desconocía su existencia, y por supuesto su contenido, pero que tú me supiste contar y aplicar para que nuestra plataforma llegara a funcionar. Gracias también a otros compañeros del HCTLab, Susana y Ángel. También he sentido vuestro apoyo en este empeño y gracias además por haber servido de revisores de la versión final. Javier ya lo trabajó bastante pero siempre queda algo que se escapa, pero ahí está Ángel para detectar hasta la inexistente coma más oculta o esa explicación que queda un poco enrevesada. Gracias Alberto y Fernando, que seguro que algún día también os lié para algún experimento y alguna aclaración software.

Gracias a los proyectandos que se involucraron también con esta idea y aportaron desarrollos, pruebas y experimentos, como Ismail, Víctor, Fernando Soto y el recién llegado Chema.

Una de las ventajas (por decir algo) de haber tardado un poco más de lo habitual en escribir la tesis es que he tenido la oportunidad de comprobar la cantidad de gente que se preocupa por mí y me asaltaban en todo momento animándome a terminarla. Todos lo hacían con cariño y aprecio, intentado incluso que no me molestara. Y nada más lejos de la realidad. Espero no haberos puesto nunca ninguna mala cara, porque siempre os lo he agradecido. Aunque alguna vez no me gustara oírlo, teníais razón. Seguro que me dejo algún nombre, y pido disculpas por ello, pero quería daros las gracias a todos: Juana, Pilar, Álvaro, Pepe, Javi, Sergio, Gustavo, Eloy, Paco, Ortega, Jorge, Doroteo, Chema, Jesús, María José, Esther, Xavier, Eduardo, Kostadin, Antonio (el *granaíno*) y un largo etcétera. Y me vais a permitir tener un recuerdo muy especial a Javier Martínez. Todavía me resuenan esas palabras suyas, "qué vamos a hacer con este chico …", repetía una y otra vez con resignación.

Gracias a mi familia. Aunque más que gracias quizás debiera pedirles disculpas. Parte del tiempo que he dedicado a este trabajo, sobre todo los últimos dos meses, se lo he quitado directamente a ellos. A Marta, mi mujer, que por cierto es la principal responsable de que yo terminara la carrera, con su apoyo, explicaciones y escondiéndome los Asterix para que me centrara en el libro correcto. Ha tenido que hacerse cargo de todas las tareas familiares, incluyendo la atención a nuestros tres hijos, y eso es de agradecer. A mis hijos, Teresa, Guille y Clara, os debo alguna visita a la feria y unos días más de vacaciones, os lo compensaré.

Y por último a mis padres, Antonio y Teresa, a quienes dedico este trabajo, por lo mucho que trabajaron y se esforzaron en unos momentos muy duros para que tanto mi hermano como yo pudiéramos estudiar. Nada me hubiera gustado más que pudieran verme ahora.

Sinceramente, gracias a todos.

Guillermo González de Rivera Peces

Resumen

Resumen

En la actualidad, construir robots es una tarea complicada que requiere especialistas en múltiples disciplinas, como hardware, software, física y mecánica. Al mismo tiempo, hay mucha investigación realizada por ingenieros software relacionados con la robótica, en campos como la Inteligencia Artificial, trabajo distribuido o sistemas colaborativos. Normalmente, prueban sus ideas con simuladores.

El objetivo principal de este trabajo ha sido diseñar un sistema para controlar de forma sencilla un robot genérico con un conjunto específico de sensores y actuadores conectados. El sistema diseñado permite trabajar sobre entornos inteligentes, centrándose en problemas concretos de la aplicación de usuario, ocultando lo demás. Se ha diseñado e implementado un conjunto de reglas, estructuras y protocolos, con flexibilidad y sencillez, denominado GdRBot, que permiten a un usuario no especializado el montaje y control de un conjunto de robots, adaptado a sus necesidades, en el que probar los algoritmos diseñados. Con esto, se facilita al usuario la capacidad de crear una gran variedad de aplicaciones, permitiendo al mismo tiempo reutilizar cualquiera de los elementos empleados en aplicaciones precedentes.

La arquitectura software de GdRBot está especialmente diseñada con el objetivo de facilitar la tarea a usuarios sin interés en el diseño de hardware y sin conocimiento de los sistemas de comunicación. La arquitectura está formada por tres tipos de componentes: clientes, servidores y elementos de red que pueden ser ejecutados en diferentes plataformas hardware. El servidor está instalado en los robots y los clientes pueden instalarse en el propio robot o en cualquier tipo de sistema ordenador. El servidor es el encargado de realizar la gestión sobre el hardware y el cliente hace uso de ese hardware para lograr una tarea. El cliente realiza solicitudes y supervisa los servidores a través de llamadas XML-RPC que se transmiten por medio de los elementos de una red TCP/IP.

La arquitectura hardware que se puede diseñar y controlar desde el sistema GdRBot permite la construcción de un conjunto de robots móviles equipados con sensores y

actuadores así como un conjunto de mecanismos de comunicación entre ellos, lo cual les permite interaccionar a modo de agentes para llevar a cabo tareas colaborativas, sin importar el medio físico sobre el que realizan la comunicación entre ellos.

También se ha definido un elemento hardware, denominado Servidor Sensorial, cuyo objetivo es reducir la carga de trabajo en el procesador central, resolviendo los accesos de bajo nivel a los diferentes sensores y actuadores que se puedan conectar a la plataforma. Se conecta al procesador central a través de un puerto de comunicaciones (serie, USB, etc) y tiene instalado un pequeño sistema operativo propio que básicamente es un servidor de comandos.

Como parte de esta tesis, y para validar su funcionamiento, se ha desarrollado un Servidor Sensorial propio, al que se ha denominado GP_Bot. Éste está formado a su vez por tres partes, la tarjeta GP_Bot que es una placa de desarrollo de propósito general basada en un micro-controlador de 8 bits, la tarjeta GP_Ifaz diseñada para actuar como interfaz entre la tarjeta GP_Bot y cierto tipo de sensores y actuadores y la tarjeta GP_Mon utilizada para la carga y depuración de programas en el micro-controlador, forzando la entrada del micro-controlador en el modo monitor.

Para el usuario final, el sistema es transparente tanto en el hardware como en el software que conforman el agente robot que se quiere controlar. Tan sólo “ve” un conjunto de funciones de alto nivel que puede llamar desde su aplicación y que realizan las tareas correspondientes en el robot. El sistema diseñado no depende de ninguna plataforma ni elemento hardware específico, por lo que se puede garantizar su longevidad en el futuro. Gracias a la generalidad de las especificaciones de diseño, se facilita la portabilidad de una determinada aplicación a otra arquitectura diferente.

Como prueba de estas capacidades, en este trabajo se han implementado cuatro arquitecturas hardware dispares entre sí y se han realizado distintos experimentos para su caracterización, basadas todas en el sistema GdRBot.

Como prueba de funcionalidad del sistema, se ha probado aplicando distintos algoritmos para tareas de posicionamiento, navegación y procesado de imágenes, desarrollando en su caso, como parte de este trabajo, aquellos drivers que han sido necesarios.

Abstract

Abstract

Today, building robots is a complex task, and specialists in multiple disciplines are required, including hardware, software, physics and mechanics. However, there is much work done by software engineers related with robotics, in fields such as Artificial Intelligence, distributed work or collaborative systems. Given the great difficulty that involves building their own platforms for this type of specialist, all these activities are ussually tested with simulators.

The main objective of the study was to design a system capable of controlling in an easy way a generic robot with a specific set of connected sensors and actuators. The designed system allows working over intelligent environments, focusing on particular problems of the user´s application and disregarding the others. A set of rules, structures and protocols called GdRBot has been designed and implemented, allowing the assembly and control of a set of robots by a non-specialist user, in a flexible and easy way, adapted to his needs, in which the designed algorithms can be tested. This system provides users the ability to create a variety of applications, while allowing reuse of any element used in previous applications.

GdRBot software architecture is specifically designed with the aim of facilitating the task to users not specially skifull in hardware design or without enough knowledge of communication systems. The architecture consists of three types of components: clients, servers and network elements that can be executed in different hardware platforms. The server is installed in the robots and the clients can be installed in the robot itself or in any type of computer system. The server is in charge of managing the hardware and the client uses that hardware to accomplish a task. The client makes requests and supervises the servers through XML-RPC calls that are transmitted by means of the elements of a TCP/IP network.

The hardware architecture that can be designed and controlled from the GdRBot allows the fabrication of a set of mobile robots equipped with sensors and actuators, as well as a set of communication mechanisms among them. This enables them

interacting like agents to carry out collaborative tasks independently the physical medium over which the communication among them is made.

A hardware element, called Sensorial Server, has been defined as well. It is aimed at reducing the workload on the central processor, solving the low-level access to the different sensors and actuators that can be connected to the platform. It is connected to the central processor via a communication port (serial, USB, etc.) and has installed its own small operating system that it is, basically, a command server.

As a part of this thesis, and with the aim of validating its operation, an own Sensorial Server, named GP_Bot, has been developed. It consists of three parts, the GP_Bot card that is a general purpose development board based on an 8-bit micro-controller, the GP_Ifaz, designed to act as an interface between the GP_Bot card and certain type of sensors and actuators, and the GP_Mon card, used for the loading and debugging of programs in the micro-controller, allowing the micro-controller to work in monitor mode.

For the end-user, the system is transparent both in hardware and software that constitute the robot agent to be controlled. The user only “sees” a set of high level functions that can be called from the application and perform the associated tasks in the robot. The designed system neither depends on any platform nor specific hardware element, so that sustainability in the future can be ensured. Thanks to the generality of the design specifications, the portability of a given application to another different architecture is facilitated.

As evidence of these capabilities, in this work four different hardware architectures have been implemented and various experiments have been performed for its characterization, all of them based on the GdRBot system.

As proof of functionality, the system has been tested by applying different algorithms for positioning, navigation and image processing tasks, developing when necessary, as part of this work, those drivers that have been required.

Índice

Índice

Lista de Figuras

Capítulo 1. Introducción

C.1.

1.1. Objetivo

Ingenieros de diferentes especialidades, investigando en diferentes campos de aplicación de la robótica, vuelcan todo esfuerzo en encontrar respuestas creativas a los problemas con que se enfrentan. En su trabajo, tienen que compatibilizar conjuntos muy amplios de restricciones, que van desde el conocimiento técnico a las necesidades de la ejecución práctica, reglas generales, compromisos económicos, disponibilidad de recursos, etc.

En esta labor tan compleja, los diseñadores cuentan cada vez más con la ayuda de un gran número de aplicaciones de software dedicado y simuladores, que les permiten conocer con bastante rapidez el comportamiento futuro de los objetos diseñados antes de que estén operativos y optimizar sus funcionalidades principales. A pesar de su potencial, estas herramientas no se pueden utilizar durante todas las fases de trabajo, llegando una etapa en la que es necesario contar con un dispositivo o robot real donde probar dichos desarrollos.

En la actualidad, construir robots es una tarea complicada y está destinada a especialistas en hardware. Sin embargo, hay mucho conocimiento generado por especialistas software que se puede aplicar a la robótica, campos como por ejemplo la Inteligencia Artificial, trabajo distribuido, redes neuronales, sistemas colaborativos, etc. Todo este trabajo se prueba ahora con simuladores por la falta de plataformas robóticas estándar.

Es muy útil, por lo tanto, un sistema hardware/software, que sustituya los simuladores de forma sencilla, controlado por un lenguaje de programación de alto nivel. La idea principal reside en que exactamente el mismo código, las mismas estructuras, los mismos algoritmos que se prueban en el simulador se pudieran instalar, de forma totalmente transparente, en una plataforma real.

El objetivo principal de este trabajo es diseñar un sistema desde el que controlar de forma sencilla un robot genérico específico, según los sensores y actuadores

conectados. Para ello, se describe una plataforma funcional sobre la que poder trabajar sobre entornos inteligentes centrándose en problemas concretos de la aplicación de usuarios, olvidándose de los demás, pues ya están implementados y funcionando en el sistema de partida.

1.2. Descripción general del sistema

Desde un punto de vista más amplio se puede ver el sistema como un punto de encuentro entre los distintos equipos de investigación, pues permite compartir desarrollos de una forma sencilla, para así crecer entre todos. A todo esto se suma que la plataforma en su totalidad está basada en estándares y herramientas libres. De esta manera todos los desarrollos realizados para esta plataforma se podrán portar a futuras plataformas abiertas de forma sencilla.

La arquitectura que se presenta pretende establecer un modelo de referencia para la rápida implementación y evaluación de sistemas multiagente. Para llegar a este objetivo se necesita una infraestructura de robots que cumplan una serie de requisitos mínimos para poder soportar el paradigma de agentes autónomos. En este sentido se debe contar con servicios de comunicación para transmitir su actividad, percepción del entorno para estimular las reacciones y movimiento para navegar por el entorno. Una vez disponibles estos servicios a bajo nivel se pueden invocar a alto nivel desde algoritmos complejos de planificación, inferencia, IA, cooperación, etc.

Este trabajo presenta una nueva plataforma robótica. Se trata de un conjunto de elementos o módulos de control, electrónicos, de software, de comunicaciones, etc..., que se podrán configurar de forma que sirvan de plataforma hardware para la implementación de los modelos neuronales de control y aprendizaje. Su estructura está enfocada a la interacción de forma autónoma con el entorno y al aprendizaje de una diversidad de tareas en el ámbito industrial y doméstico.

Una plataforma como la descrita se ha de componer de un robot lo suficientemente genérico como para poder realizar múltiples tareas. Así mismo, el robot ha de poder comunicarse con el exterior de varias formas para poder funcionar en múltiples escenarios. Por último, se ha de tener una librería de alto nivel para comunicarse con el robot desde los programas que diseñen los especialistas en software. Por supuesto,

además debe de ser suficientemente flexible para poder incorporar cualquier nuevo recurso sin necesidad de modificar el existente ni el firmware que lo soporta.

El robot es la parte más importante de la plataforma, ha de ser capaz de realizar la mayor cantidad de tareas posibles y permitir que se le añadan de forma simple dispositivos para realizar otras tareas no pensadas en un principio.

De acuerdo con lo anterior, las funcionalidades que debe ofrecer esta plataforma son las siguientes:

- ✓ Acceso al Hardware mediante rutinas simples. Uno de los objetivos del presente trabajo es facilitar la tarea a los especialistas en software sobre el control del hardware. Para ello, dicho control se encapsula dentro de unas rutinas muy optimizadas a bajo nivel, ya escritas por especialistas en diseño y desarrollo hardware, que cumplen una sencilla interfaz para ser llamadas desde estructuras de alto nivel.

  Ejemplos de este tipo de rutinas pueden ser:

  - Desplazamiento de un punto a otro determinado, indicando qué robot se quiere mover, la dirección a seguir y la velocidad. También se programan los valores de aceleración y deceleración. Para conseguir llegar al punto deseado hace uso de la información de los *encoders* (sensores de desplazamiento) de los motores, de forma implícita. Por ejemplo:

    int **mover_trapezoide**(lr_robot *robot,BYTE direccion, SIGNED_32 posicion, SIGNED_32 velocidad, UNSIGNED_32 aceleracion, UNSIGNED_32 deceleracion);

  - Desplazamiento, indicando el robot a mover, la dirección a seguir y la velocidad. También se programan los valores de aceleración y deceleración. En este caso no se hace uso de la información de los encoders. Por ejemplo:

    int lr_mover_velocity(lr_robot *robot,BYTE direccion, SIGNED_32 velocidad, UNSIGNED_32 aceleracion, UNSIGNED_32 deceleracion);

- ✓ Agentes Móviles (Robots) económicos, extensibles y abiertos. Al realizar un diseño independiente del hardware, se pueden utilizar equipos muy accesibles, sin

grandes requisitos de funcionamiento e incluso utilizando equipamiento del que ya se dispone. La forma de comunicación entre los diferentes sistemas también está completamente definida, por lo que resulta muy sencillo el añadir nuevos sistemas, bien que estuvieran previstos, bien que sean propios y exclusivos del desarrollo que se esté haciendo en ese momento.

- ✓ Adición de nuevo hardware mediante una sencilla interfaz (Servidor Sensorial). La plataforma está concebida como una unidad de proceso central a la que se añaden diferentes funcionalidades a través de sistemas independientes, denominados Servidores Sensoriales. Como se verá posteriormente con más detalle, este mecanismo es el que permite añadir más elementos al conjunto de una forma totalmente transparente.

- ✓ Intercomunicación de los agentes desde prácticamente cualquier medio físico. Otro de los principales objetivos de la plataforma es crear sistemas colaborativos basados en diferentes agentes, para lo cual es imprescindible dotar de sistemas de comunicación a dichos agentes. La plataforma soporta cualquier medio físico que sea capaz de gestionar una pila TCP/IP (Ethernet, Wifi, puertos serie, paralelo, bluetooth, GSM, GPRS, etc…).

- ✓ Acceso al sistema desde cualquier sistema operativo/lenguaje de programación. La plataforma está concebida como un todo, un conjunto de elementos totalmente funcionales de forma autónoma, no necesitan ningún elemento del exterior. Tiene un único punto de acceso a través del cual se le comunican las órdenes que debe ejecutar y a ese punto de acceso se puede acceder desde cualquier sistema operativo y desde cualquier lenguaje de programación. Son un conjunto de órdenes precisas y definidas, conocidas por el usuario de la plataforma, cuyas llamadas se pueden invocar desde cualquier tipo de plataforma, basada en cualquier sistema operativo ejecutando código de cualquier lenguaje de programación elegidos todos por el usuario final y no por el diseñador de la plataforma.

- ✓ Fácil integración con la web. Esta es una conclusión lógica que emana del hecho citado en uno de los puntos anteriores: la conectividad se basa en TCP/IP, es decir, cada agente se puede ver directamente como un servidor web.

1.2.1. Hardware de la plataforma

La arquitectura inicial elegida está basada en PC, puesto que es una arquitectura muy probada, ampliamente extendida y con gran cantidad de dispositivos.
A la hora de extender las capacidades del robot se ha elegido una interfaz tipo USB, debido a su gran versatilidad y la gran cantidad de dispositivos que se pueden encontrar en el mercado.

Para la comunicación con el exterior, se ha querido dar soporte al máximo posible de elementos de red tales como:

- Ethernet
- Wireless
- Puerto paralelo
- Puerto serie
- Puerto usb
- Bluetooth

Físicamente el robot está formado, en líneas generales, por:

- ✓ Unidad Central de Control. Sistema embebido que contiene el microprocesador, centro de la inteligencia del sistema. Este sistema embebido ejecutará el sistema operativo apropiado para llevar cabo todas las tareas de gestión y de comunicación. Incluye una tarjeta Wireless que permita una comunicación Ethernet inalámbrica, básica para el intercambio de información con otros agentes.

- ✓ Servidores Sensoriales. Son pequeños elementos de gestión de sensores y actuadores que, dotados de un microcontrolador, permite liberar a la Unidad Central de Control de la ejecución de ciertas tareas más simples, permitiendo

incluso una concurrencia de tareas al poder ejecutarlas en diferentes sistemas. Está documentada la interfaz con la unidad de control, de forma que el propio usuario puede conectar servidores sensoriales desarrollados por él mismo. Deben poder controlar elementos como:

- Sistema locomotriz. Los robots pueden ser móviles, por lo que se dota de un mecanismo de locomoción para su desplazamiento, así como la electrónica necesaria para su correcta gestión.
- Cámara de video. Para gestión de imágenes.
- Micrófono. Procesamiento de audio, ejecución de tareas por voz.
- Altavoz. Interacción con el exterior.
- Pantalla LCD. Presentación de resultados, valores de configuración, estado, mediciones, etc.
- Sensores de posición y distancia. Permiten situar al robot en el entorno, dibujar un mapa de objetos que lo rodea, evitar obstáculos para poder evitarlos, etc.

1.2.2. Software del robot

El robot ha de contar con un sistema operativo lo suficientemente potente y flexible como para poder controlar todo el hardware antes mencionado. Así mismo ha de estar ampliamente orientado a la comunicación a través de redes, para facilitar la conectividad del robot con todos los medios físicos posibles.

El sistema ha de ser de un tamaño reducido, pues sería deseable que funcionara en una memoria flash de pequeño tamaño. Además, sería muy deseable que fuera abierto, para poder realizar todas las modificaciones que necesitemos al mismo de forma sencilla sin recurrir a la ingeniería inversa.

Bajo el sistema operativo se ejecuta un programa servidor, que se ha denominado Servidor Robótico. Atiende las peticiones de los clientes y accede a los elementos conectados a través de los Servidores Sensoriales.

1.2.3. Acceso al robot

A la hora de establecer una comunicación con el robot, se ha de ofrecer un sistema genérico, accesible desde cualquier arquitectura y desde cualquier sistema operativo. De esta forma no se obliga al investigador que utilice la plataforma a cambiar sus herramientas de desarrollo, podrá seguir trabajando exactamente igual que lo hacía antes. Se ha elegido XMLRPC, pues está formado por XML sobre http y a su vez sobre TCP/IP, todo ello muy estándar y simple. El Servidor Robótico recibe y atiende las peticiones XML-RPC.

1.3. Estructura final

Como resultado final del trabajo de tesis, se ofrece a los usuarios una serie de elementos de los que disponer para probar sus trabajos, representados en la figura 1.1, y son los que componen el Sistema GdRBot. Estos son:

- Elementos software:
    - Servidor Robótico. Programa que actúa como servidor y se ejecuta sobre el sistema operativo Linux. Incluye la interfaz de acceso a todas las funciones que se pueden invocar desde el cliente, a través de XML-RPC.

- Elementos Hardware:
    - Unidad Central de Control. Sistema basado en microprocesador, con los recursos necesarios, tanto de potencia de cálculo como de gestión de periféricos, así como conectividad con el exterior tipo Ethernet o Wifi.
    - Servidor Sensorial. Elementos de gestión y control de sensores y actuadores. Incluyen un gestor de comandos, con una interfaz común para todos, que permite el acceso a sus funciones por parte de cualquier unidad de control que conozca dicha interfaz.

El usuario final tan sólo debe decidir, en función de las necesidades de la aplicación y los recursos de los que dispone, qué partes son las que necesita adquirir y cuáles las aporta él mismo.

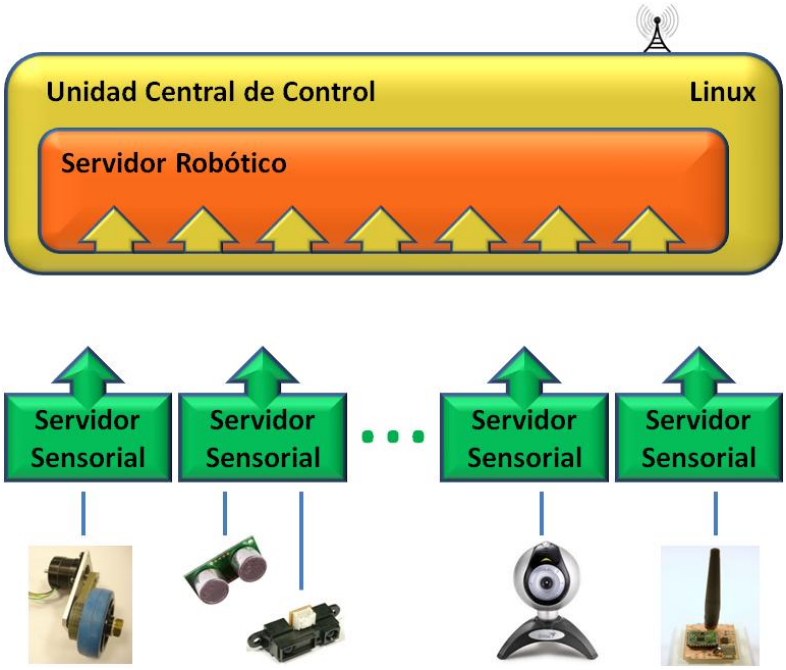


Figura 1.1. Estructura del Sistema GdR-Bot

Con esta decisión se pueden crear tres escenarios diferentes:

Escenario 1. El usuario no aporta nada. En este caso, tras documentar las necesidades de la aplicación que pretende desarrollar, deberá adquirir lo siguiente:

- Unidad Central de Control. Además de la arquitectura hardware necesaria, tendrá instalada la distribución de Linux correspondiente y el Servidor Robótico, compilado para esa distribución y la arquitectura del procesador utilizado. Estarán documentadas todas las funciones y servicios implementados en ese servidor. Incluye hardware y software.
- Servidores Sensoriales. Habrá tantos Servidores Sensoriales como necesite para acceder a los sensores y actuadores de los que quiera disponer. Incluye hardware y software así como documentación de la interfaz con la unidad central de proceso.

Escenario 2. El usuario aporta la plataforma donde instalar el Servidor Robótico (un viejo ordenador portátil, una placa de un PC sobrante, un sistema de desarrollo que quedó de un proyecto anterior, etc). En este caso lo que deberá adquirir es:

- Aplicación software del Servidor Robótico. Tras instalar Linux en su plataforma, compilará el código fuente del Servidor Robótico para la distribución que haya instalado y para su plataforma.
- Servidores Sensoriales. Análogo al escenario 1.

Escenario 3. El usuario aporta la plataforma y tiene los conocimientos y medios necesarios para desarrollar sus propios servidores sensoriales. Lo que debe adquirir es lo siguiente:

- Aplicación software del Servidor Robótico. Tras instalar Linux en su plataforma, compilará el código fuente del Servidor Robótico para la distribución que haya instalado y para su plataforma.
- Documentación de la interfaz de comunicación entre el servidor sensorial y la unidad de control. De esta forma, programará el gestor de comandos del servidor sensorial para mantener un diálogo correcto.

En cualquier escenario de los anteriores siempre existe la opción de o bien ampliar el número de sensores y/o actuadores, sin más que añadir los Servidores Sensoriales correspondientes o bien incluso cambiar la unidad de control si ésta pasa a no tener los recursos necesarios.

Capítulo 2. Estado del arte en robótica desde el punto de vista de agentes móviles cooperativos

C.2.

2.1. Introducción

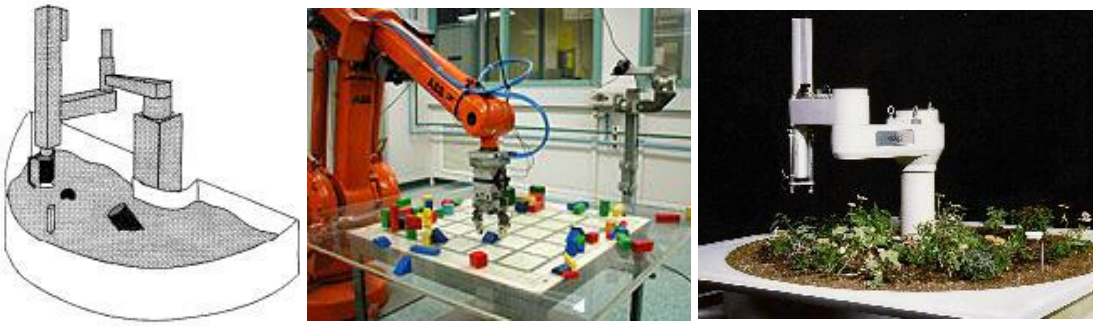
El objetivo de este capítulo es presentar un estudio sobre el estado del arte en robótica desde el punto de vista de agentes autónomos cooperativos. Este estudio ha consistido en la búsqueda de conceptos que posteriormente se deben poder aplicar en la plataforma a diseñar, conceptos tales como *cooperación, elementos colaborativos, agentes físicos, scheduling y dualidad entre sistemas centralizados vs sistemas distribuidos*, siempre en contextos relacionados con robótica. Se ha considerado de vital importancia la realización de este estudio previo por la formación de base del autor de esta tesis.

2.2. Conceptos generales

2.2.1. Telecontrol vía web

En el trabajo de Golderg et al. [2.1] realizado en 1995 ya aparece el concepto de **telecontrol mediante HTTP** donde se describe el *"Mercury Project"*, basado en el control vía web de un robot manipulador equipado con una cámara CCD móvil y dotado de un sistema neumático capaz de emitir un chorro de aire comprimido. Posiblemente haya sido éste el primer robot tele-operado a través de la web. Poco después de la publicación en Internet del *"Mercury Project"* en septiembre de 1994, Ken Taylor publica el robot UWA [2.2]. Este robot, en su origen, consistía en un brazo con dos grados de libertad, situado dentro de un recipiente con arena que tenía objetos enterrados. El objetivo era que los operadores remotos encontraran esos objetos. Posteriormente, del mismo autor, este proyecto evoluciona en el *"Telerobotic Garden"* [2.3], en donde los usuarios remotos del sistema pueden ver un jardín e incluso interactuar con él. Éstos pueden plantar semillas, regar y comprobar el progreso de crecimiento todo a través del control de movimiento de un brazo robot. En el trabajo de P. Saucy et al. [2.4] los operadores deben sacar al robot controlado de un laberinto. En estos experimentos los operadores remotos colaboran tomando el

control de manera secuencial. En [2.5] se propone la **colaboración simultánea** a través de Internet ejemplificada mediante un juego de la Ouija. En [2.6] se describe un sistema multi-operador y multi-robot en la que los robots tele-operados cooperan en actividades tales como resolver un cubo de Rubik.



Mercury Project [2.1] UWA[2.2] Telegarden[2.3]

Figura 2.1. Proyectos pioneros en tele-operación web

En [2.7] se aborda el problema del **retardo** en la tele-operación, y en [2.8] se destaca la dificultad de ejercer un control preciso de un robot a través de Internet, por las deficiencias inherentes a este medio, comparando el control a través de Internet del robot inalámbrico MAX al control que se ejerce sobre un perro mediante la correa, al indicarle órdenes mediante tirones y donde la precisión de la respuesta depende de factores externos al operador como puede ser el interés del perro en seguir otra trayectoria.

2.2.2. Plataforma software

En el trabajo de Golberg et al. [2.1] la plataforma software está formada por tres subsistemas intercomunicados denominados *Server A*, *Server B* y *Server C*. *Server A* consiste en un servidor web NCSA ejecutado en un sistema UNIX que atiende las peticiones de los usuarios vía HTTP (interpreta las coordenadas ISMAP generadas al marcar los usuarios con el ratón un punto de la imagen generada por la cámara y sirve la imagen más reciente del robot) y gestiona la cola de usuarios autorizados. *Server B* es un servidor de base de datos para la autenticación de usuarios corriendo sobre la misma máquina que *Server A*. *Server C* controla el robot a través de un interfaz serie. *Server A* y *Server C* se comunican mediante TCP/IP sobre una red Ethernet.

En [2.9] se propone una plataforma software de tres capas: un cliente Java, JC (*Java Client*), una plataforma de servicio de robot en red, NRSP (*Net Robot Service Platform*) y un controlador de robot, RC (*Robot Controler*). NRSP da servicio a JCs y RCs, aceptando conexiones de múltiples operadores y múltiples robots de manera concurrente. La comunicación entre NRSP y RCs se lleva a cabo mediante CORBA sin perjuicio en el rendimiento, puesto que RCs y NRSP están conectados a través de LAN (*Local Area Netwok*). Para la comunicación entre NSRP y JCs se utilizan sockets TCP/IP mediante un *applet* Java descargable con un navegador web. NRSP gestiona la interacción entre JCs y RCs actuando de servidor *proxy* para ambos.

La plataforma software que se presenta en este trabajo de tesis [2.11] está formada por dos tipos de elementos: servidores y clientes. Hay un servidor en cada robot atendiendo las peticiones de los clientes que pueden residir en cualquier sistema hardware dotado de un procesador, incluidos los propios robots (en este caso para realizar tareas de supervivencia o de cooperación con otros robots). La arquitectura permite realizar acciones de tres tipos: supervivencia, control remoto e interoperabilidad. Para la comunicación entre clientes y servidores se utiliza XML-RPC sobre HTTP a nivel de aplicación (mediante un servidor web Apache), TCP/IP para el transporte y numerosas posibilidades de capa física (USB, Bluetooth, puerto serie, puerto paralelo, GSM/GPRS, radio y WLAN). El sistema operativo para los servidores es Linux. Internamente los servicios están programados en lenguaje C. El cliente es una aplicación programada en cualquier lenguaje para el que se disponga de una librería robótica encargada del intercambio de mensajes XML-RPC, con independencia del sistema operativo.

Con posterioridad a la fase inicial de este trabajo en 2006 apareció la primera versión comercial de Microsoft Robotics Studio [2.10]. Se trata de una herramienta basada en Windows para la creación de aplicaciones robóticas. Está orientado al desarrollo modular de aplicaciones en diferentes niveles de abstracción con respecto al hardware y permite realizar simulaciones. Incluye un modelo de programación aplicable a diferentes plataformas hardware, favoreciendo la reutilización de tecnología.

A diferencia de la plataforma propuesta en esta tesis, sólo permite interactuar con robots basados en PC y que estén ejecutando el sistema operativo Windows. El

desarrollo de las aplicaciones sólo es en lenguaje C o en VPL (*Microsoft Visual Programming Languaje*), lo que obligará al usuario a un nuevo aprendizaje.

La plataforma de Microsoft hace una propuesta comercial en la misma línea que la marcada por el objetivo de este trabajo de tesis. Conviene manifestar que los primeros trabajos publicados fruto de la línea de investigación emprendida se publicaron en diversos congresos en 2005, [2.11], [2,12] y [2.13], en fechas anteriores a la presentación de la herramienta de Microsoft.

2.2.3. Navegación

Una referencia frecuente en lo que respecta al concepto de robot autónomo es el robot-guía de museo como por ejemplo en la plataforma MINERVA [2.14]. A través de esta plataforma, se abordan temas relacionados con la navegación de robots tales como localización, generación de mapas, elusión de colisiones y planificación de recorridos.

2.2.3.1. Localización y generación de mapas

En el trabajo de Thrun et al. [2.15] se considera la localización de robots móviles como un problema de estimación probabilística según el método de máxima verosimilitud. Propone un modelo probabilístico para el movimiento y otro para la percepción. Puesto que el movimiento del robot se asume impreciso, la posición del robot tras ejecutar una orden de movimiento desde una posición determinada se define mediante una función de densidad de probabilidad; esto es, el robot va acumulando errores de rotación y traslación según se desplaza. En cuanto a la percepción, se asume que el robot es capaz de interpretar su entorno en cuanto a la estimación de tipo, ángulo relativo y distancia aproximada a puntos de referencia, modelando así un mapa. Sin embargo esta interpretación puede no ser del todo correcta, de manera que la percepción de un punto de referencia se modela como una función de densidad de probabilidad condicionada a la bondad del mapa. La obtención de los mapas se resuelve mediante el método de máxima verosimilitud a partir de los datos estimados. Puesto que el método puede ser computacionalmente inabordable, se recurre al algoritmo EM (*Expectation-Maximization*), que es un método de escalada (*hill-*

climbing) basado en modelos de Markov, así como a diversas técnicas para mejorar la eficiencia (*caching*, probabilidades simétricas, baja resolución temporal en la estimación de posición y cálculos y memorización selectiva)

En [2.16] se propone un método basado en la monitorización constante de la incertidumbre de la posición estimada y la re-localización automática del robot cuando la incertidumbre es alta, interrumpiendo la actividad normal del robot si es preciso y volviendo a ella una vez efectuada la corrección. Para ello se utiliza localización de Markov y se obtiene una estimación de la incertidumbre de la misma. Si la localización resulta ser poco fiable, el robot pasa a una re-localización activa (consistente, por ejemplo, en desplazarse hacia una posición que permita obtener la localización de manera unívoca). Para determinar la acción apropiada para re-localizarse se tratan de minimizar la incertidumbre de la futura posición y el coste de la ejecución.

2.2.3.2. Elusión de colisiones y planificación de ruta

En [2.17] se describe el método de elusión de colisiones de ventana dinámica DWA (*Dinamyc Window Aproach*). Este mecanismo se caracteriza por tener en cuenta los parámetros dinámicos del movimiento del robot (tales como la inercia) de manera que la búsqueda de posibles comandos de control del robot para el instante de tiempo siguiente se realizan directamente en el espacio de velocidades. La búsqueda está acotada por las restricciones dinámicas (velocidades alcanzables en el instante siguiente) y por la seguridad (velocidades seguras con respecto a los obstáculos). En primer lugar se reduce el espacio de búsqueda y en segundo lugar se escoge de entre las velocidades posibles aquella que maximice la función objetivo. La reducción del espacio se realiza en tres pasos: (a) Se consideran únicamente trayectorias circulares determinadas por pares (u,w) de velocidad de rotación y traslación, resultando en un espacio de búsqueda bi-dimensional. (b) La restricción de velocidades admisibles asegura que solamente se considerarán trayectorias seguras. Un par (u,w) se considera admisible si el robot es capaz de frenar antes de alcanzar el siguiente obstáculo para la curvatura correspondiente. (c) La ventana dinámica reduce las velocidades admisibles a aquellas alcanzables en un intervalo corto de tiempo teniendo en cuenta las limitaciones de aceleración del robot. En cuanto a la optimización, se maximiza la

función objetivo mediante un compromiso entre los siguientes aspectos: (a) medida del progreso hacia la posición del objetivo, (b) distancia al obstáculo más próximo en la trayectoria y (c) velocidad del robot.

En [2.18] se plantean las limitaciones de la elusión de colisiones basada estrictamente en sensores y se propone un mecanismo híbrido denominado mDWA (*mapping DWA*) que combina el mecanismo DWA con un mapa del entorno. La posición relativa del robot en el mapa se obtiene mediante localización de Markov. Mediante este mecanismo es posible evitar obstáculos indetectables mediante los sensores disponibles, siempre y cuando su posición en el mapa sea conocida.

En [2.19] se presenta un método de elusión de obstáculos para un robot guía que combina DWA con otros dos métodos: banda elástica [2.20] y NF1 [2.21]. Este método busca la suavidad del movimiento para comodidad de los visitantes guiados por el robot. DWA se encarga de generar comandos para los actuadores de modo que el robot no colisione, las posibilidades dinámicas de los actuadores no sean violadas y el robot se mantenga en la banda elástica. La banda elástica se responsabiliza de la representación de la trayectoria, adaptando el plan al movimiento del robot y a los cambios en el entorno. NF1 es responsable de generar el plan inicial.

La generación de mapas de colisión es un elemento interesante a tratar en este estudio bibliográfico, por considerar que resuelve problemas que aparecen cuando hay más de un agente o robot circulando, y además son factibles de ser generados entre los implicados por medio de trabajo colaborativo, funcionalidad que se desea incorporar en la plataforma a diseñar.

En el trabajo de Seung-Hwan y Beom-Hee [2.22], se propone un método de elusión de colisiones entre robots basado en "mapas de colisión" para la generación de rutas seguras que resumimos a continuación. Señalar que las figuras están extraídas de la citada referencia [2.22].

Mapas de colisión

Si se consideran dos robots: 1 y 2, con radios respectivos R1 y R2, utilizando el espacio de obstáculos se puede representar el robot 1 como el robot con radio R1+R2 y considerar al robot 2 como un punto, figura 2.2. El robot 1 tiene la máxima prioridad,

de modo que no modifica su trayectoria. Por el contrario el robot 2 debe modificar su trayectoria si existe alguna posibilidad de colisión. Se asume por simplicidad que las trayectorias son líneas rectas.

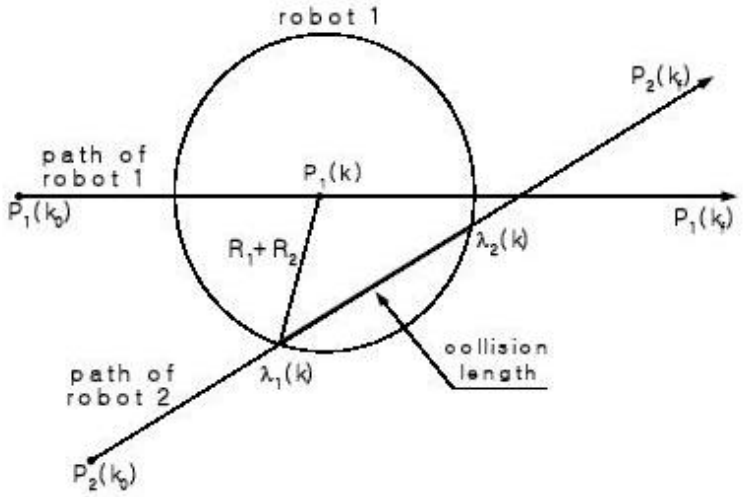


Figura 2.2. Trayectoria de dos robots y longitud de colisión. [2.22]

El segmento formado por la intersección de la trayectoria del robot 2 con el robot 1 (de radio R1+R2) se denomina longitud de colisión y se examina cada instante del intervalo muestreado. El conjunto de longitudes de colisión obtenidas forman una región de colisión.

Si la TLVSTC (*traveled length versus servo time curve*) del robot 2 entra en esta región, figura 2.3, indica que se producirá una colisión según las trayectorias originales

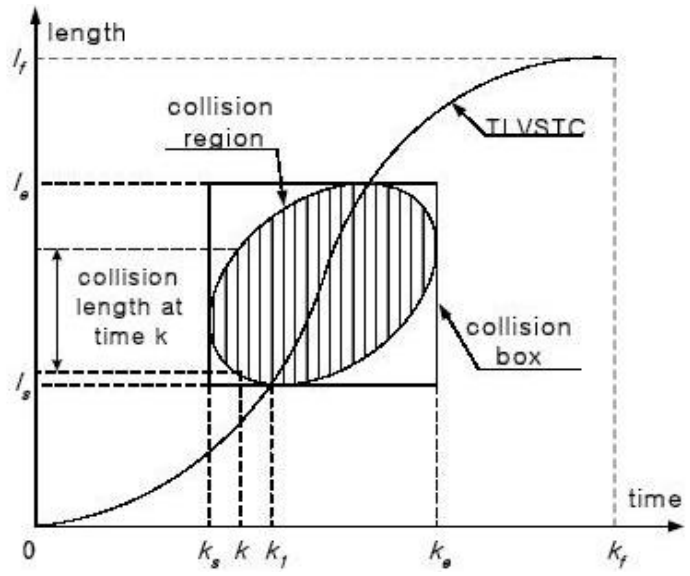


Figura 2.3. TLVSTC, región y cuadrado de colisión. [2.22]

Esta colisión puede ser analizada algebraicamente, dando lugar a tres tipos de solución: i) puede no tener soluciones reales (en cuyo caso no hay colisión), ii) puede tener una solución doble y real (correspondiéndose con el momento en que el robot 1 está entrando o saliendo de la trayectoria del robot 2), o iii) dos soluciones reales (invadiendo el robot 1 la trayectoria del robot 2, con lo que existe posibilidad de colisión).

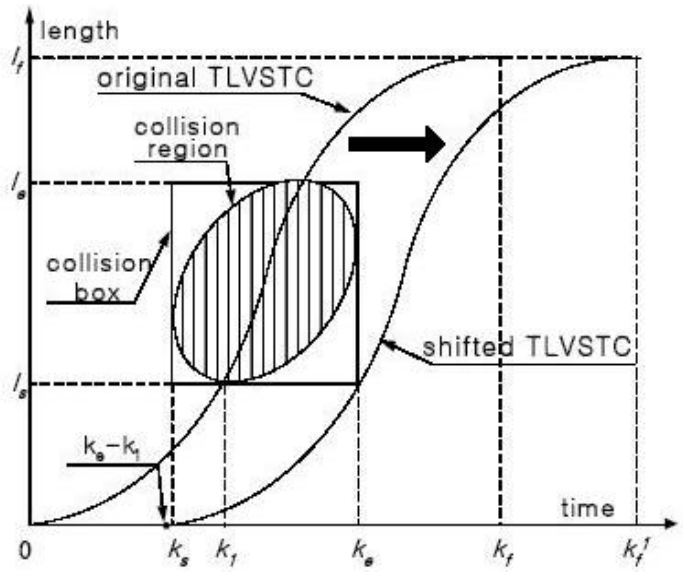


Figura 2.4. Elusión de colisión mediante retardo de tiempo. [2.22]

Para evitar la colisión la TLVSTC del robot 2 no debe entrar en la región de colisión. Para simplificar matemáticamente la representación de la región de colisión, se utiliza un cuadrado de colisión y se obtienen las coordenadas de sus límites, que se utilizan para modificar la trayectoria del robot 2. La modificación de la trayectoria se realiza principalmente mediante dos técnicas: retardo de tiempo y reducción de velocidad. En la primera, figura 2.4, el inicio del movimiento del robot 2 se retrasa K_e-K_1 en la figura 2.5. En la segunda, figura 2.5, los robots comienzan a moverse a la vez, y se modifica la velocidad del robot 2. Utilizando la segunda puede darse el caso de que la velocidad del robot 2 se termine reduciendo hasta cero, de modo que esta técnica presenta un rendimiento más bajo que la de retardo de tiempo en lo que a tiempo en alcanzar el objetivo se refiere.

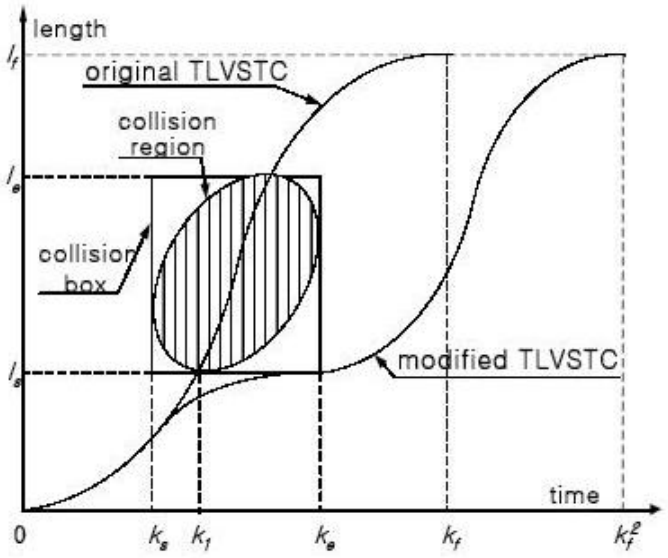


Figura 2.5. Elusión de colisión mediante reducción de velocidad. [2.22]

Traslación de la región de colisión

Se considera ahora la elusión de colisión de un robot en términos de traslación de la región de colisión, que se corresponde en realidad con la traslación de su trayectoria. Como antes, el cuadrado de colisión hará las veces de la región de colisión por simplicidad. Se clasifican los desplazamientos en 4 casos (derecha, izquierda, abajo y arriba; en este orden en la figura 2.6). Se siguen asumiendo trayectorias en línea recta. En el mapa de colisión se puede denotar cualquier trayectoria por el parámetro l entre 0 y 1 independientemente de la forma. Aquí, desplazar cada región de colisión de modo que no invada la TLVSTC evita las colisiones de forma obvia.

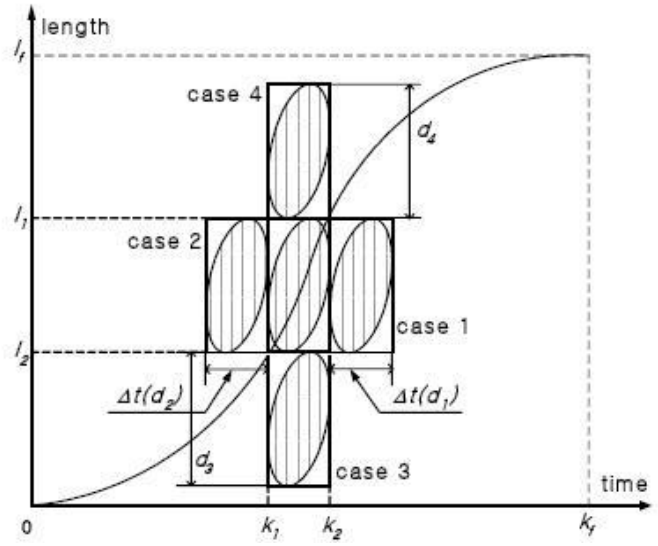


Figura 2.6. Traslaciones de la región de colisión. [2.22]

El desarrollo matemático que apoya este razonamiento, así como detalles de aplicación del área de colisión puede consultarse en [2.22].

2.2.4. Sistemas Modulares

Entre las referencias encontradas en [2.23] se identifica el concepto de sistema n-modular en PolyBot [2.9] y [2.24], un sistema n-modular y auto-reconfigurable

En [2.24] se repasan algunos modelos de PolyBot. PolyBot consta de un solo módulo activo denominado *segment* con un grado de libertad y dos superficies de conexión. Cada módulo dispone de un microprocesador, así como de numerosos sensores (de contacto, aceleración, etc). Se dice de los sistemas n-modulares que prometen ser versátiles (pueden adoptar diferentes formas), fiables (pueden auto-repararse desprendiéndose de módulos defectuosos) y barato (al estar formados por múltiples elementos idénticos son aplicables las economías de escala). PolyBot fue el primer robot modular en conseguir modificar su modo de desplazamiento mediante auto-reconfiguración, pasando desde una configuración en anillo y desplazamiento de oruga a una configuración lineal y desplazamiento de serpiente. La auto-reconfiguración consistió en un simple desprendimiento (*detach*) de un par de módulos.

2.2.5. Clasificación de los sistemas modulares

En el trabajo de González-Gómez et al. [2.23] se propone una nueva clasificación de sistemas modulares. A la clasificación existente formulada por Mark Yim, en la que se diferencian los robots en retículo y en cadena se añaden tres subgrupos de configuración en cadena: 1D, 2D y 3D. Si el robot está formado por una serie de módulos dispuestos en cadena, entonces se dice que es una cadena 1D. Dos o más cadenas se pueden conectar formando topologías 2D sobre el plano (como triángulos, cuadrados y estrellas). Finalmente, las cadenas pueden conectarse de modo que no se puedan disponer sobre un mismo plano, formando topologías 3D como cubos o pirámides, figura 2.7.

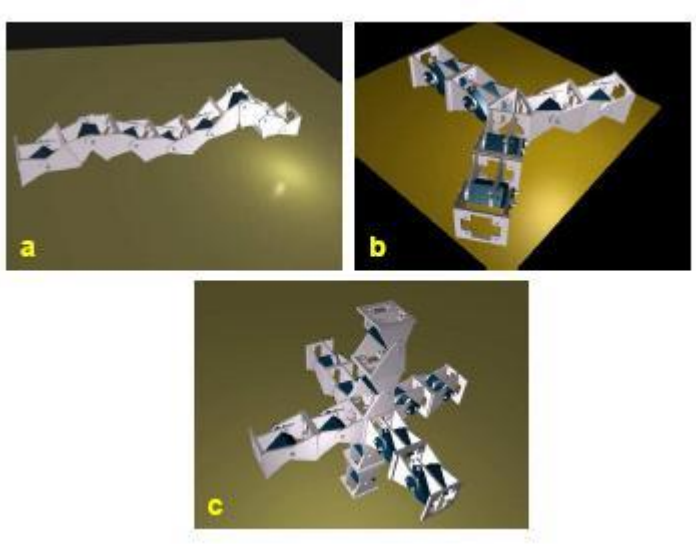


Figura 2.7. Clasificación de robots n-modulares.

La familia de robots 1D son apropiados para atravesar tuberías, asir objetos o moverse por terreno irregular. Con suficiente longitud pueden formar un bucle y desplazarse como una rueda. En general son más estables puesto que tienen mayor superficie de contacto con el suelo. Pueden dividirse en dos grupos: *serpentine* y serpiente. Los primeros disponen de ruedas o propulsores y los segundos se desplazan mediante movimientos corporales.

Los robots serpiente se pueden dividir a su vez en tres subgrupos atendiendo al eje de conexión de los módulos adyacentes: *pitch*, *yaw* y *pitch-yaw,* figura 2.8. El desplazamiento en configuración *yaw* es, como el de las serpientes al arrastrarse, tangencial al eje del cuerpo. Diversos estudios referenciados en este artículo tratan este tipo desplazamiento. El desplazamiento en configuración *pitch* se realiza en una dimensión, mediante ondas que atraviesan el cuerpo del robot desde la cola a la cabeza. El tipo de movimiento conseguido depende de parámetros como la amplitud, frecuencia o longitud de la onda. Puede efectuar una sencilla auto-reconfiguración formando un bucle y moviéndose como una rueda. En configuración *pitch-yaw*, algunos módulos rotan alrededor del eje *pitch* y otros alrededor del eje *yaw*. Estos robots consiguen nuevos modos de desplazamiento, como rotar o rodar.

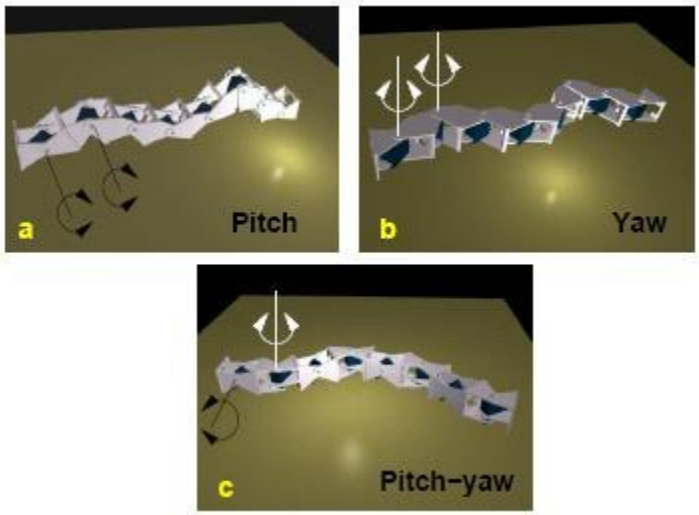


Figura 2.8. Clasificación de robots n-modulares.

2.3. Cooperación y elementos colaborativos

2.3.1. Arquitecturas colaborativas

En [2.25] se diferencian los grupos de robots en homogéneos y heterogéneos en función de si las habilidades de sus miembros son o no las mismas. Se describen algunas arquitecturas representativas de colaboración como

- **CEBOT**: arquitectura descentralizada, jerárquica y celular, basada en la organización celular de algunas entidades biológicas. El sistema es reconfigurable dinámicamente en cuanto a que los robots como células básicas autónomas que pueden acoplarse físicamente a otras, reconfiguran su estructura para obtener una óptima configuración en respuesta a cambios en el entorno. Existen **células maestras** que coordinan sub-tareas y se comunican entre sí.
- **ACTRESS**: arquitectura en la que un grupo heterogéneo de actores se coordinan para realizar una tarea que de otro modo no podría ser llevada a cabo individualmente por ninguno de los actores.
- **SWARM**: sistema distribuido con un conjunto homogéneo y numeroso de robots autónomos no inteligentes que exhiben un comportamiento colectivo

inteligente. La auto-organización dentro de un SWARM es la habilidad para distribuirse de manera óptima para realizar una tarea determinada. La interacción se produce mediante reacción de cada célula al estado de sus vecinas.

- **GOFER**: la arquitectura GOFER se utilizó para estudiar la resolución distribuida de problemas por múltiples robots mediante técnicas de inteligencia artificial. Utiliza una sistema centralizado de planificación que se comunica con los robots y tiene una visión global del problema y de la disponibilidad de los robots que han de realizar las tareas. Los robots utilizan un algoritmo de asignación de tareas para obtener su misión.
- **ALLIANCE**: arquitectura desarrollada para el estudio de la cooperación en grupos heterogéneos, de tamaño pequeño o mediano, de robots autónomos independientes y con pocas similitudes. Se asume en los robots la capacidad de tener consciencia del efecto de sus acciones y de las de los demás agentes mediante percepción y comunicación explícita mediante difusión. Los robots individualmente actúan según un controlador basado en comportamiento con una extensión para activar conjuntos de comportamientos para desempeñar ciertas tareas. Estos conjuntos son activados por comportamientos motivacionales cuya activación se determina a su vez en el robot por la percepción de sus compañeros. Existe una extensión a esta arquitectura, denominada L-ALLIANCE, que utiliza aprendizaje de refuerzo para ajustar los parámetros de activación de los conjuntos de comportamientos.
- **Comportamiento cooperativo basado en comportamientos**: se basa en la síntesis de comportamientos colectivos tales como la agrupación, la búsqueda de alimento o el acoplamiento, basados en comportamientos primitivos básicos como son huida, seguimiento, dispersión o establecimiento de hogar.

2.3.2. Comunicación

En [2.26] se pone de manifiesto la fragilidad impuesta al establecer una jerarquía en un sistema colaborativo multi-agente, destacando tres motivos: que la comunicación entre el maestro y el enjambre de agentes esclavos puede suponer un cuello de

botella, que la posibilidad de perder al maestro en entornos peligrosos con la consiguiente reorganización resta robustez al sistema y que la necesidad de diseño de elementos diferentes (maestros, esclavos o agentes válidos para ambos roles) supone una complejidad añadida. Propone la validez de un sistema multi-agente no jerárquico para entornos peligrosos, de comportamiento similar al de una colonia de insectos buscando alimento, al que denomina **cooperación sin comunicación**. Los agentes buscan individualmente y cuando varios agentes encuentran un mismo objetivo cooperan en su transporte en ausencia total o parcial de comunicación.

En [2.25] se destaca que la estructura de comunicación de un grupo determina los posibles modos de interacción inter-agente, y se caracterizan a grandes rasgos tres tipos de interacción:

- **Interacción a través del Entorno** donde el propio entorno es el medio de comunicación, sin que exista interacción ni comunicación explícita entre agentes (este es el tipo descrito en [2.26])
- **Interacción Sensorial** basada en la interacción local entre agentes que se detectan mutuamente (mediante infrarrojos, radio o visión) sin existir comunicación explícita. Se emplea, por ejemplo, para establecer formaciones de robots.
- **Interacción mediante Comunicación** en la que se produce una comunicación explícita entre agentes mediante mensajes intencionados, bien sea dirigidos a agentes concretos o por difusión.

2.3.3. Coordinación

En el trabajo de Sweene et al. [2.27] se analiza la restricción **L.O.S.** (*Line of Sight*) como sub-objetivo en un sistema de comunicación de entornos distribuidos. Según esto, dos robots coordinados tratan de controlar la distancia que les separa manteniendo una mutua visibilidad. Existe un *leader* que se dirige al objetivo fijado y un *follower* que sigue los pasos del *leader*. Se plantean dos configuraciones: *push* y *pull*. En configuración *pull* el *leader* se dirige al objetivo, perseguido por el *follower* que se mantiene dentro de la región LOS. El *follower* puede permanecer quieto hasta que la

restricción LOS amenace con ser incumplida o bien seguir siempre de cerca los movimientos del *leader*. Atendiendo a esto el seguimiento puede ser agresivo, neutro o conservador. En configuración *push* es el *follower* quien marca la trayectoria del *leader* "empujándole" a desplazar la región LOS.

En la figura 2.9 se puede ver un ejemplo gráfico de restricción LOS. El robot 0, que es el *leader*, tiene como objetivo alcanzar el punto oscuro situado en la esquina inferior izquierda. El robot 1 adopta un comportamiento *pull* agresivo como *follower* de 0, mientras que los robots 2 y 3 adoptan un comportamiento *pull* más conservador como *followers* de 1 y 2, respectivamente, y el robot 4 permanece estacionario.

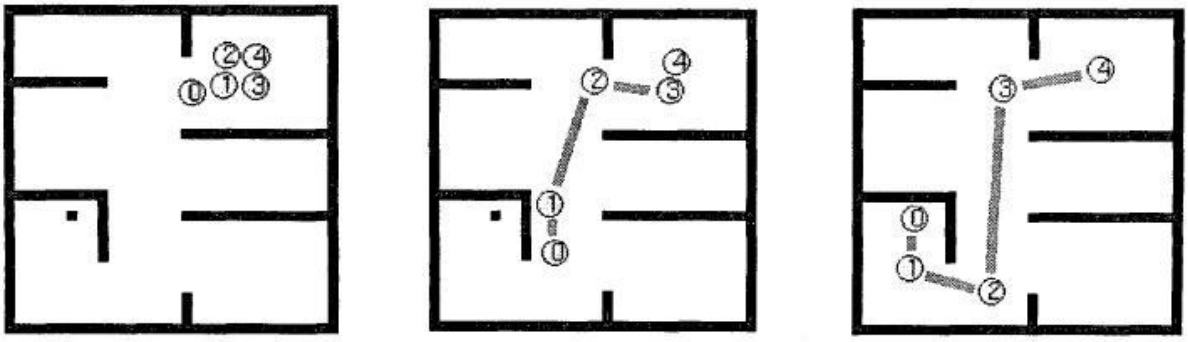


Figura 2.9. Ejemplo de restricción LOS y comportamiento "pull".

2.3.4. División en celdas

En el trabajo de Larionova et al. [2.28] se describe un sistema de sensores olfativos mediante el cual un equipo de robots se coordina para no limpiar dos veces la misma zona. El artículo propone una división del área total en celdas que se puedan cubrir con movimientos hacia adelante y hacia atrás (esto se consigue mediante descomposición *boustrophedon* tratada en Choset et al. [2.29]). Como consecuencia, las celdas cubiertas lo están completamente. Así, cuando un robot detecta rastros de otro al entrar en una celda, la abandona inmediatamente asumiendo que ya ha sido cubierta. La detección se realiza mediante dos narices electrónicas, evitando falsos positivos (debidos por ejemplo a corrientes de aire) mediante estimación.

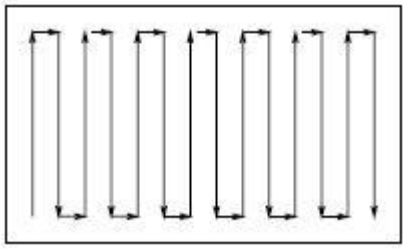


Figura 2.9. Trayectoria boustrophedon.

2.3.5. Planificación *(scheduling)*

En [2.30] se analiza el problema de la asignación de tareas dentro de un sistema multi-robot. Existen *n* trabajadores buscando tarea, hay *m* tareas a realizar y cada una requiere un trabajador. Cada tarea puede tener un nivel de prioridad diferente, y cada trabajador un nivel de adecuación diferente para una determinada tarea. El objetivo es asignar trabajadores a tareas maximizando el rendimiento global, teniendo en cuenta las prioridades y los niveles de adecuación. Para la estimación de la adecuación de un robot a una tarea en el artículo se define un indicador de **utilidad** que es función de la calidad de ejecución esperada, dados el procedimiento y el equipamiento a utilizar, y del coste estimado del recurso, dado los requerimientos espacio-temporales de la tarea. Se comparan 6 arquitecturas diferentes en cuanto a su complejidad computacional y de comunicación: ALLIANCE [2.31], BLE [2.32], M+ [2.33], MURDOCH [2.34], *First-Price auctions* [2.35] y *Dynamic Role Assignment* [2.36]

En [2.37] se considera la planificabilidad (*schedularity*) como requisito fundamental para la escalabilidad. Presenta una implementación distribuida de un controlador coordinado para comportamiento *leader/follower* y restricción LOS.

En [2.38] se describe un algoritmo para encontrar una estrategia factible de entre un conjunto de estrategias funcionalmente equivalentes en la distribución de tareas en un equipo coordinado de robots para alcanzar el objetivo de una aplicación. El algoritmo se basa en el paradigma de cadena de robots que mantienen la línea de visión (LOS) tratado en [2.27] en la que cada par de robots se divide en *leader* o *follower,* actúan según esquema *pull* o *push*, y tratan de minimizar el coste de comunicación y los recursos de procesador utilizados. Una estrategia se considera factible si y solo si dentro del mínimo común múltiplo de los períodos de tarea (en el artículo las tareas se consideran periódicas) cada instancia de tarea es programada para comenzar y

completarse dentro de los plazos establecidos, además de satisfacer las restricciones impuestas como, por ejemplo, la precedencia de tareas. Para encontrar una estrategia factible el sistema debe asignar tareas a procesadores libres minimizando el coste de comunicación y la carga de cada procesador y determinar una programación factible para todas las tareas, incluyendo las de comunicación. El *leader* decide qué estrategia es la adecuada en cada momento y la transmite al equipo. La estrategia se reconsidera periódicamente en previsión de cambios en la topología o en el tamaño del equipo. Se analizan dos algoritmos de asignación: el ambicioso (*greedy*) y el agresivo. En ambos se intenta asignar al mismo procesador tareas que se comunican entre sí, evitando de este modo el coste de comunicación.

2.3.6. Agentes físicos

El concepto de Agente Físico (*Physical Agent* o PAS) lo introduce Mizukawa en [2.39] basándose en el trabajo de Sheridan [2.40] y [2.41] sobre control de manipuladores. El PAS que propone trata de un sistema robot autónomo y tele-operado que hace las veces de Agente en tareas controladas remotamente por personas. Está compuesto por un robot móvil (mediante orugas) autónomo equipado con una cámara CCD, un puntero láser y un brazo articulado. Se puede controlar a través de diversos mecanismos como joystick, guante sensorial, órdenes vocales o un simulador 3D VRML (*Virtual Reality Modeling Language*), y es también accesible a través de una PDA. Como utilidad más inmediata, Mizukawa propone la utilización de su PAS en operaciones de rescate en entornos peligrosos y para el cuidado de personas mayores y discapacitados.

En [2.42] se propone un robot guardia de seguridad comunicado a través de telefonía móvil CDMA (*Code Division Multiple Access*).

2.3.7. Sistemas centralizados vs distribuidos

En [2.25] se trata este concepto como arquitectura de grupo. Se clasifican los sistemas en centralizados (con un solo agente de control) y descentralizados (sin agente de control), y estos últimos en jerárquicos (que son localmente centralizados) y distribuidos. Se destaca que pese a que se ha tratado ampliamente la superioridad de

los sistemas descentralizados, no se ha encontrado ninguna publicación en la que se establezca una comparativa empírica o teórica al respecto. Por otro lado muchos sistemas no encajan estrictamente a un lado de la dicotomía centralizado/descentralizado (por ejemplo, muchas arquitecturas descentralizadas utilizan agentes *leader*)

En [2.43] se aboga por arquitectura distribuida para la navegación de sistemas móviles, en la que las decisiones se toman por votación. Un conjunto de comportamientos (seguridad, elusión de obstáculos, seguimiento de la carretera, consecución del objetivo, etc) votan la acción a realizar, y un árbitro escoge la más votada. Combinando adecuadamente comportamientos planificados con reactivos se puede conseguir que se complementen y compensen sus deficiencias.

En [2.44] se describe un sistema de impresión multi-robot utilizando dos arquitecturas diferentes: una centralizada y otra descentralizada. En el experimento realizado se comparan ambas arquitecturas. La descentralizada muestra mayor tolerancia a fallos y flexibilidad sin planificación previa. La centralizada muestra un mejor rendimiento en tiempo de ejecución

2.4. Agentes, agentes inteligentes, multiagentes

La Inteligencia Artificial (IA) se caracteriza por ser un área de trabajo especialmente interdisciplinario, en la que se pueden distinguir dos grandes campos de trabajo. El primero de ellos estaría involucrado en la creación de los elementos hardware necesarios para poder estar frente a un “entorno inteligente”, tales como sistemas de comunicación, sistemas biométricos, sistemas de localización, etc. El segundo de ellos se encargaría del desarrollo de los algoritmos y aplicaciones software necesarios para dar funcionalidad al hardware anteriormente mencionado.

Esta amplia interdisciplinaridad, necesaria para crear sistemas que ayuden a realizar las tareas diarias a los usuarios, acarrea graves problemas a la hora de desarrollarlos. Por un lado los equipos de investigación no suelen estar especializados en todas las áreas de conocimiento necesarias, lo que hace que las investigaciones acaben en meros algoritmos funcionando sobre un simulador, en hardware poco fiable o, en el peor de los casos, en meras especulaciones. Por otra parte, si el equipo se decide por

realizar el trabajo completo para el desarrollo del sistema, se enfrenta a muchos problemas en múltiples campos, lo que supone una gran infraestructura y tiempo de desarrollo.

Por su naturaleza interdisciplinaria la IA ha estado influenciada por áreas interesadas en la interacción entre grupos de individuos inteligentes. Muchas veces en la investigación el énfasis no se ha puesto en los problemas a resolver sino en los agentes que se verán implicados en la interacción que conduce a la solución de los mismos.

Un agente se puede concebir como una entidad computacional, especialmente ubicada en un entorno y que es capaz de mostrar un comportamiento flexible (varias reacciones posibles ante una misma situación e interacción con otros agentes) y autónomo (capaz de tomar la iniciativa sin estímulo previo) [2.45]. A esta noción de agente se le añade la componente de movilidad intrínseca de los robots, que inmediatamente implica navegación.

Debido a la gran cantidad de definiciones diferentes encontradas, a continuación se muestra una lista de las características más habituales que aparecen en la bibliografía [2.46]:

Los agentes son entidades que relacionan *percepciones* de su entorno con *acciones* en su entorno, es decir, son *reactivos*.

Son objetos computacionales con un grado variable de *autonomía*

- Los agentes toman *iniciativas*, es decir, no actúan simplemente en respuesta a su entorno sino que pueden actuar en base a unos objetivos propios.
- Interactúan y se comunican mediante un lenguaje de comunicación que incorpora *actos de habla* (*speech acts*) como *informar*, *preguntar*, *prometer* y *requerir*.
- Son racionales, es decir, actúan de “forma correcta", lo cual se interpreta como que el estado mental del agente es *coherente* con su comportamiento.

Estas características de los agentes pueden sugerir diferentes modelos, de mayor o menor complejidad. Por ejemplo, en informática, la idea de agente como un proceso software que se ejecuta de forma concurrente, que tiene una noción de estado y es

capaz de comunicarse con otros agentes por pase de mensajes, es algo que se considera como un resultado natural del desarrollo del paradigma de programación concurrente orientada a objeto. Esta noción de agente es la que también se usa en la llamada "ingeniería de software basada en agentes", cuyos productos, llamados "*software agents*" o "*softbots*", son agentes que interaccionan con un entorno software mediante comandos que usan para obtener información o cambiar el estado del entorno.

En cambio, dentro de la IA, la noción de agente parece adecuarse más a un sistema computacional que además se conceptualiza usando nociones mentales, como pueden ser las nociones de *conocimiento*, *creencia*, *intención* u *obligación*. Así pues, dado que nuestro interés sobre los agentes se centra en el ámbito de la IA y los robots móviles autónomos y cooperantes, añadiremos a la lista anterior la siguiente característica:

- Los agentes son unidades que se pueden describir en términos de *estados mentales*: conocimientos, intenciones, creencias, deseos, etc...

Conceptualizar los agentes no es tarea fácil, por eso se ha optado por definir un conjunto de propiedades o atributos que caracterizan a los agentes aunque esto no implica que todos las posean:

- *Autonomía*: Operan sin intervención de otros agentes para encontrar sus objetivos diseñados, además tienen alguna clase de control sobre sus acciones y estados internos.
- *Sociabilidad* (*comunicación*): Interaccionan con otros agentes (humanos o no), utilizando para ello un lenguaje de comunicación entre agentes. Pueden comunicarse además con varios recursos del sistema o usuarios, de allí que los que interactúan directamente con el usuario se llamen asistentes personales o agentes de interfaz. Desde el punto de vista del agente los recursos pueden ser locales o remotos, hay un amplio rango de sistemas de recursos de agentes que pueden ser accedidos, por ejemplo programas de aplicación, bases de datos, sistemas de información, etc.
- *Cooperación*: Permiten la cooperación entre entidades de agentes, la complejidad de la cooperación puede variar desde un estilo de interacción

cliente/servidor a negociaciones y cooperación basada en métodos de inteligencia artificial, tales como redes de contrato y protocolos. Esta cooperación puede necesitar del intercambio de información y representaciones de prerrequisitos para sistemas multiagentes.

- *Reactividad:* Perciben estímulos de su entorno (el mundo físico, un usuario vía una interfaz gráfica, una colección de otros agentes, Internet o todos ellos combinados) y reaccionan ante ellos posiblemente para cambiar lo que allí ocurre.
- *Proactividad/Iniciativa*: Tienen carácter emprendedor y actúan guiados por sus objetivos. También ésta propiedad se puede referir como *orientado a objetivos.*
- *Movilidad*: Se trasladan a través de una red telemática para desempeñar tareas específicas. Generalmente se puede identificar dos niveles de movilidad de los agentes:
    - *Ejecución remota*: Un agente es transferido a un sistema remoto donde es activado y ejecutado en su totalidad, el mecanismo de transporte del agente utilizado varía desde TCP/IP a correo electrónico.
    - *Migración*: Durante su ejecución un agente activo puede moverse de nodo a nodo para cumplir progresivamente su tarea, en otras palabras el agente puede suspender su ejecución, transportarse el mismo a otro nodo de la red y reanudar la ejecución desde el punto en el cual fue suspendida, además pueden lanzar nuevos agentes durante su viaje, por ejemplo para adquirir información para su cliente o para ejecutar subtareas específicas.
- *Veracidad*: No comunican información falsa a propósito (se supone).
- *Benevolencia:* Ayuda a otros agentes y no entra en conflicto con sus propios objetivos.
- *Racionalidad:* Actúa en forma racional con miras a cumplir sus objetivos.
- *Inteligencia:* Se refiere al método utilizado para desarrollar la lógica del agente o la inteligencia y está estrechamente relacionada con los lenguajes de agentes

donde predominan dos aspectos: la creación de contenido pragmático del agente y la representación del conocimiento que proporciona los medios para expresar objetivos, tareas, preferencias y vocabulario apropiado para varios dominios.

- *Adaptativo (Aprendizaje):* Cambia su comportamiento basado en las experiencias previas.
- *Carácter:* Se puede creer que tienen personalidad y estados emocionales.
- *Operación Asíncrona*: El agente puede ejecutar tareas totalmente desacoplado de sus usuarios o de otros agentes, lo que significa que puede ser disparado por la ocurrencia de un evento particular.

La noción de comportamiento consiste en una reacción programada ante un estímulo predeterminado. Sirve para dotar al robot de reacciones rápidas básicas ante eventos muy comunes y definidos ("pared"$\rightarrow$"frenar") de manera que se libera a las tareas de alto nivel (planificación, IA, etc) de su procesamiento.

Una de las principales líneas en el campo de la robótica está dedicada a la emulación de los modelos biológicos de comportamiento en los sistemas artificiales. La puesta en marcha de estas técnicas se ha enfocado desde dos puntos de vista. Por un lado, la implementación de modelos de inteligencia artificial en *sistemas robóticos antropomórfico*s, que tienden a imitar en el mayor grado posible los dispositivos sensoriales, motores y de articulación de la estructura física del ser humano.

Por otro lado, ha aparecido lo que ciertos autores han denominado agentes autónomos. Una definición para este tipo de dispositivos puede ser la de "un sistema situado dentro y como parte de un entorno, que siente y nota su entorno y actúa en él en el tiempo para cubrir su propia agenda, siendo capaz de apreciar los resultados obtenidos y de volver a actuar, tomándolos en consideración" [2.47]. Se trata de dispositivos robóticos con cierta inteligencia y capacidad de interactuar con el mundo real y entre ellos.

Existe una diferencia bien clara entre los robots antropomórficos y los agentes autónomos. En los primeros, el objetivo fundamental es conseguir el mayor grado de emulación de los sistemas biológicos en cuanto a comportamiento, sistemas de

percepción a modo de dispositivos oculares, táctiles, de vibración, deslizamiento, esfuerzo en la realización de tareas, etc. Uno de los ejemplos más ilustrativos se puede encontrar en los robots bípedos o humanoides [2.48]. Por estos motivos los robots antropomórficos precisan de un gran esfuerzo a nivel de desarrollos hardware, electrónicos y de adquisición, siendo éste aún el objeto de innumerables trabajos de investigación.

En cuanto a los agentes autónomos, la filosofía es completamente distinta. Se trata de dispositivos en los que el diseño antropomórfico no aporta un valor añadido importante. Por el contrario, la característica principal de tales dispositivos es la de conseguir una capacidad de aprendizaje, de comportamiento a nivel tanto individual como de grupo e incluso de evolución y supervivencia a partir de modelos genéticos. Es evidente que el esfuerzo fundamental en este caso no es el desarrollo hardware sino funcional, es decir la capacidad para realizar un elevado número de tareas a partir de su aprendizaje, bien sea por refuerzo o por imitación.

Los agentes autónomos robóticos tienen la capacidad de interactuar entre ellos, pudiendo realizar tareas tales como jugar un partido de fútbol [2.49].

Una simplificación o restricción de los agentes autónomos son los denominados *real-world artifacts* [2.50]. Se trata de robots autónomos e inteligentes que actúan en un entorno real, bajo circunstancias de incertidumbre pero que a diferencia de aquéllos, carecen de una capa de control para comportamiento en grupo con otros robots. Los robots así configurados, están dotados de determinados dispositivos que los hacen muy útiles dentro del entorno doméstico, donde la interacción entre robots no se plantea como una necesidad.

Los agentes físicos incorporan a los agentes software la problemática de la interfaz numérico-simbólica y simbólico-numérica que caracteriza los sistemas físicos que, de acuerdo a [2.51], están sometidos a restricciones del mundo real tales como imprecisión, incertidumbre, variabilidad en el tiempo, entre otras características. Una realización común, aunque no es la única, de los agentes físicos son los robots móviles a los que se les incorpora en la investigación actual una alta autonomía y capacidad de trabajar de forma cooperativa.

La IA tradicional ha estado principalmente buscando la metodología para manipular símbolos usados en la adquisición y representación de conocimiento, y su razonamiento, gastando poca o nula atención en sus aplicaciones a mundos dinámicos reales. Mientras, en robótica se ha puesto mucho énfasis en los temas de diseño y construcción de hardware y su control. Actualmente, los tópicos recientes cubren dos áreas que incluyen los principios del diseño de agentes autónomos, colaboración multi-agente [2.52], adquisición de estrategias, razonamiento en tiempo real y planificación, robótica inteligente, *sensor-fusion* y aprendizaje de comportamientos. Estos temas exponen nuevos aspectos que son difíciles de resolver con los enfoques tradicionales.

Para la solución de estos nuevos aspectos y finalmente alcanzar el objetivo último de la IA y la robótica consistente en construir sistemas inteligentes que resuelven de forma emergente tareas complejas en un mundo real dinámico (aquí *dinámico* entendido como un mundo altamente cambiante, nuevos escenarios, nuevos objetos, etc.), los cuerpos físicos tienen el rol importante de conducir el sistema agente a una interacción significativa con el entorno físico, complejo, incierto, pero que es regido por un conjunto de restricciones. El significado de "tener un cuerpo físico" puede resumirse como sigue:

i. Las capacidades sensoriales y las de actuación están altamente ligadas.

ii. Para que el agente ejecute las tareas encomendadas, los espacios sensoriales y de actuación deben ser abstraídos dentro de los recursos del agente (memoria, potencia de proceso, controladores, etc.).

iii. Dicha abstracción depende de las interacciones del agente con el entorno.

iv. Fruto de la abstracción, cada agente tiene un modelo de representación del entorno.

v. En el mundo real, tanto las interacciones inter-agentes como agentes-entorno son asíncronas, paralelas y arbitrariamente complejas. Los agentes físicos evolucionan de forma continuada a tiempo continuo.

vi. La complejidad natural de las interacciones físicas genera distribuciones fiables de datos para los algoritmos de aprendizaje de los agentes físicos, en comparación a los agentes software.

Los *retos* que en investigación deben estar en continuo avance y que no siempre están resueltos, en el caso de agentes físicos vienen expresados en [2.53] y se pueden resumir en:

- Percepción. Los agentes físicos deben tener capacidad de usar una percepción de amplio rango, discriminar otros agentes físicos, estimar sus posiciones y la suya propia, entre otras. La percepción es una aplicación básica que se extiende en las aplicaciones robóticas.
- Acciones. El agente físico ha de ser capaz de controlar su cuerpo físico. En el caso de robots móviles, ha de ser capaz de ejecutar trayectorias, seguir objetivos, los cuales no han de ser necesariamente estáticos.
- Situación y comportamiento. Aunque las tareas a realizar sean simples, casi infinitas situaciones aparecen debido a los continuados cambios del entorno, como por ejemplo los objetivos móviles, que otros agentes físicos se desplacen en el mismo entorno, etc.
- Tiempo real. Como que la situación cambia continuamente, hay restricciones de tiempo en la toma de nuevas decisiones, re-planificación de trayectorias, y adecuación de los controles.
- Plataforma. Debe decidirse cuál es la plataforma de experimentación de los agentes físicos que facilite la investigación internacional e interdisciplinaria sobre dichos agentes.

En cuanto a los sistemas multi-agente, históricamente conocido como *Inteligencia Artificial Distribuida* y dividido en dos campos; Resolución Distribuida de Problemas (DPS, *Distributed Problem Solving*) y Sistemas Multi-Agente (MAS), ahora se engloban ambos en un significado más general. *DPS* considera cómo un problema predeterminado es resuelto por un número también determinado de módulos, donde todas las interacciones son parte del sistema. En cambio los *MAS* se ocupan del comportamiento de una posible colección pre-existente de agentes autónomos

heterogéneos aspirando a resolver un problema que está por encima de las capacidades o conocimiento individuales [2.53].

Algunas características de los sistemas multi-agentes son que cada agente tiene información y capacidad incompleta para resolver el problema, no existe un control centralizado ni datos centralizados y la computación es asíncrona.

Mediante los MAS se persigue una mejora de la robustez y la eficiencia, pero ello implica una serie de retos complejos, descritos en [2.54] y algunos resueltos en [2.55]:

- Cómo formular y descomponer los problemas entre un grupo de agentes inteligentes
- Cómo diseñar los protocolos de interacción
- Cómo asegurar la coherencia de las interacciones no locales basadas en decisiones locales
- Cómo razonar sobre el estado de la coordinación de las acciones y los planes
- Cómo identificar y reconciliar intenciones o puntos de vista opuestos
- Cómo gestionar los recursos limitados entre comunicación y computación
- Cómo evitar un comportamiento conjunto inestable, caótico u oscilatorio
- Cómo idear metodologías de desarrollo y construcción práctica de MAS

El primer modelo de Multi-Agentes propuesto como estándar de concurrencia, eran básicamente componentes autónomos de un mismo sistema que se comunican mediante el paso de mensajes. Las primitivas básicas de estos actores eran: *create* (a partir de una descripción), *send* (mensaje a otro actor) y *become* (cambio de estado). Este modelo también presenta los *retos de la coherencia* sobre cómo alcanzar objetivos no-locales de alto nivel tan sólo con conocimiento local [2.56].

Ya desde comienzos de la historia de la IA Distribuida, se comenzó a pensar en la asignación flexible de tareas. Gracias a Davis y Smith [2.57] y posteriormente a Sandholm [2.58] se desarrolló el protocolo *Contract Net*, que consiste en hacer que los agentes tomen dinámicamente uno de los roles: o bien *manager*, o bien *contractor.* Dada una tarea, el agente la examina si se puede dividir en subtareas para poder

ejecutarlas concurrentemente. El *manager* ofrece la tarea a subasta entre todos los *contractors*, y estos según el estado de sus recursos ofrecerán una oferta indicando el grado de capacidad con el que cree que puede completar esa tarea. En un principio fue concebido como un protocolo de negociación, pero ahora se considera de coordinación. A pesar de conseguir un balanceo natural de la carga de trabajo, existen algunos defectos en el protocolo original: el *manager* no informa a los nodos qué ofertas han sido rechazadas, no existe interrupción de las tareas (que den paso a tareas críticas), hace un uso intensivo del sistema de comunicaciones, etc.

2.5. Referencias del Capítulo 2

[2.1] "Desktop teleoperation via the World Wide Web", K. Goldberg, M. Mascha, S. Gentner, N. Rothenberg, C. Sutter, J. Wiegley, en Proc. IEEE Int. Conf. on Robotics and Automation (ICRA'95), vol. 1, pp: 654-659. 1995.

[2.2] "Australia's telerobot on the web (The UWA Telerobot)", K. Taylor en http://telerobot.mech.uwa.edu.au/Telerobot/index.html

[2.3] "A Tele-Robotic Garden on the World Wide Web", K. Goldberg, J. Santarromana, G. Bekey, S. Gentner, R. Morris, C. Sutter, J. Wiegley en http://queue.ieor.berkeley.edu/~goldberg/garden/Ars/

[2.4] "KhepOnTheWeb: open access to a mobile robot on the Internet", P. Saucy, F. Mondada, en IEEE Robotics & Automation Magazine, Vol: 7, Issue: 1. Mar/2000.

[2.5] "Collaborative teleoperation via the Internet", K. Goldberg, B. Chen, R. Solomon, S. Bui, B. Farzin, J. Heitler, D. Poon, G. Smith, en Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '00), vol. 2, pp: 2019-2024. 2000.

[2.6] "An Internet-based distributed multiple-telerobot system", W. Xiao-Gang, M. Moallem, R.V. Patel, en IEEE Transactions on Systems, Man and Cybernetics, Part A, Vol. 33, Issue: 5. Sept. 2003.

[2.7] "Internet-based remote teleoperation", K. Brady, T. Tzyh-Jong en Proc. IEEE Int. Conf. on Robotics and Automation (ICRA'08), vol.1, pp.65-70,1998.

[2.8] "MAX: wireless teleoperation via the World Wide Web", A. Ferworn, R. Roque, I. Vecchia en IEEE Canadian Conference on Electrical and Computer Engineering, vol.3, pp 1380-1384, 1999.

[2.9] "PolyBot and PolyKinetic System: a modular robotic platform for education", A. Golovinsky, M. Yim, Z. Ying, C. Eldershaw, D. Duff, en Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '04).vol. 2, pp: 1381- 1386. 2004.

[2.10] "Microsoft Robotics Studio" en http://msdn.microsoft.com/robotics/

[2.11] "A Generic Software Platform for Controlling Collaborative Robotic System using XML-RPC", G. Glez. de Rivera, R. Ribalda, J. Colás, J. Garrido en Proc.

IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics. (AIM'05), pp. 1336-41. Monterey (USA). July/2005.

[2.12] "Plataforma genérica para desarrollos basados en agentes móviles", G. Glez. de Rivera, R. Ribalda, J. Colás, J. Garrido en Actas de Ubiquitius Computing & Ambient Intelligence (UCAmI'05). pp: 363-369. Granada. Sept/2005

[2.13] "Hardware Independent Architecture for Autonomous Colaborative Agents", G. Glez. de Rivera, R. Ribalda, K. Koroutchev, J. Colás, J. Garrido en Proc. of 2nd Int. Conf. on Informatics en Control, Automation & Robotics (ICINCO'05), pp: 459-462. Barcelona, Sept/2005.

[2.14] "MINERVA: A Second-Generation Museum Tour-Guide Robot", S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox en Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'99), vol.3, no., pp.1999-2005, 1999.

[2.15] "A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots", S. Thrun, W. Burgard, D. Fox, en Autonomous Robots. 5, 3-4, 253-271, Springer, July/1998.

[2.16] "Integrating Active Localization into High-level Robot Control Systems", M. Beetz, W. Burgard, D. Fox, A.B. Cremers en Robotics and Autonomous Systems, 23, 4, 205-220 (16). Elsevier, July/1998.

[2.17] "The dynamic window approach to collision avoidance", D. Fox, W. Burgard, S. Thrun, en IEEE Robotics & Automation Magazine, Vol: 4, Issue: 1. Mar/1997

[2.18] "A hybrid collision avoidance method for mobile robots", D. Fox, W. Burgard, S. Thrun, A.B. Cremers en Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'98) vol. 2, pp: 1238-1243, 1998.

[2.19] "Smooth and efficient obstacle avoidance for a tour guide robot", R. Philippsen, R. Siegwart en Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '03), vol.1, no., pp. 446- 451, 14-19 Sept. 2003

[2.20] "Elastic bands: connecting path planning and control", S. Quinlan, O. Khatib, en Proc. of IEEE Int. Conf. on Robotics and Automation. vol., no., pp.802-807 vol.2, 2-6 May 1993

[2.21] "Robot Motion Planning", J.C. Latombe (autor), ISBN 0792391292. Springer. 1991

[2.22] "A New Analytical Representation to Robot Path Generation with Collision Avoidance through the Use of the Collision Map" (P. Seung-Hwan, L. Beom-Hee, in Int. Journal of Control, Automation, and Systems, 4, 1, 77-86, February/2006.

[2.23] "Locomotion Capabilities of a Modular Robot with Eight Pitch-Yaw-Connecting Modules", J. González-Gomez, H. Zhang, E. Boemo, J. Zhang, en Proc. of 9th Int. Conf. on Climbing and Walking Robots (CLAWAR'06), LNCS, September 2006.

[2.24] "PolyBot: a modular reconfigurable robot" (M. Yim, D.G. Duff, K.D. Roufas, en Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '00), vol 1, pp: 514-520, 2000.

[2.25] "Cooperative Mobile Robotics: Antecedents and Directions", Y.U. Cao, A.S. Fukunaga, A. Kahng, en Autonomous Robots. Vol. 4, nº 1. Springer Netherlands. DOI: 10.1023/A:1008855018923 March/1997.

[2.26] "Cooperation without Communication: Multiagent Schema-Based Robot", R.C. Arkin en Journal of Robotic Systems 9(3), 351-364, John Wiley & Sons, Inc. 1992.

[2.27] "Coordinated teams of reactive mobile platforms", J. Sweeney, T.J. Brunette, Y. Yang, R. Grupen, en Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '02), vol 1, pp: 299-304, 2020.

[2.28] "Olfactory coordinated area coverage", S. Larionova, N. Almeida, L. Marques, A.T. de Almeida, en Autonomous Robots. 20, 3, 251-260. Springer Netherlands. June/2006.

[2.29] "Coverage Path Planning: The Boustrophedon Decomposition", H. Choset, P. Pignon, en International Conference on Field and Service Robotics, 1997.

[2.30] "Multi-robot task allocation: analyzing the complexity and optimality of key architectures", B.P. Gerkey, M.J. Mataric en Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '03). vol. 3, pp: 3862- 3868. 2003.

[2.31] "ALLIANCE: An architecture for fault tolerant multirobot cooperation". L.E. Parker en IEEE Transactions on Robotics and Automation, 14, 2, 220-240, 1998.

[2.32] "Broadcast of local eligibility for multi-target observation", B.B. Werger, M.J. Mataric en Proc. of Autonomous Agents. Pp 1232-1238, 2000.

[2.33] "M+: a scheme for multi-robot cooperation through negotiated taskallocation and achievement", S.C. Botelho, R. Alami en Proc. IEEE Int. Conf. on Robotics and Automation (ICRA'99), vol. 2, pp: 1234-1239. 1999.

[2.34] "Sold!: auction methods for multirobot coordination", B.P. Gerkey, M.J. Mataric, en IEEE Transactions on Robotics and Automation, 18, 5, 758- 768. 2002.

[2.35] "Multi-robot exploration controlled by a market economy", R. Zlot, A. Stentz, M.B. Dias, S. Thayer, en Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '02), vol 3, pp: 3016-3023., 2002

[2.36] "Dynamic role assignment for cooperative robots", L. Chaimowicz, M.F.M. Campos, V. Kumar, en Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '02), vol 1, pp: 293- 298. 2002.

[2.37] "Scalability and schedulability in large, coordinated, distributed robot systems", J.D. Sweeney, L. Huan, R.A. Grupen, K. Ramamritham, en Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '03), vol 3, pp: 4074-4079, 2003.

[2.38] "Resource management for real-time tasks en mobile robotics", H. Lia, K. Ramamrithamb, P. Shenoya, R.A. Grupena, J.D. Sweeney en Journal of Systems and Software. doi:10.1016/j.jss.2006.09.035. 2006.

[2.39] "Physical agent system: concept and implementation", M. Mizukawa, K. Endo, T. Otsuka, Y. Ando en Proc. IEEE Int. Symp. on Computational Intelligence en Robotics and Automation, 2003

[2.40] "Telerobotics, Automation and Human Supervisory Control", T.B. Sheridan, MIT Press. ISBN 0262193167. 1992.

[2.41] "Supervisory Control of Remote Manipulators, Vehicles and Dynamic Processes", T.B. Sheridan, NTIS, Springfield, VA (USA), 1983

[2.42] "Design and implementation of real-time security guard robot using CDMA networking", R. Je-Goon, S. Hyeon-Min, K. Se-Kee, L. Eung-Hyuk, C. Heung-Ho, H. Seung-Hong, en Proc. of the 8th Int. Conf. en Advanced Communication Technology (ICACT'06). Pp 3-6, 2006.

[2.43] "DAMN: A Distributed Architecture for Mobile Navigation", J.K. Rosenblatt, en Proc. of the AAAI Spring Symp. on Lessons Learned from Implememted Software Architectures for Physical Agents. Pp 339-260, 1997

[2.44] "Multi-robot cooperation-based mobile printer system", L. Kang-Hee, K. Jong-Hwan, en Robotics and Autonomous Systems. 54, 3 , 31, 193-204, March/2006.

[2.45] “Intelligent Agents: Theory and Practice”, M. Wooldridge, N.R. Jennings, en The Knowledge Engineering Review, 10:2, pp. 115-152. 1995

[2.46] “Intelligent Agents: Theory and Practice”, Wooldridge M., Jennings N.R., en The Knowledge Engineering Review, Vol 10:2, pp. 115-152. 1995

[2.47] “Is it an agent, or just a program?: A taxonomy for Autonomous Agents", S. Franklin, A. Graesser. Proc. 3 Int. Workshop on Agent Theories, Architectures and Languages, pp: 193-206. 1996.

[2.48] "The Development of Honda Humanoid Robot", K. Hirai, M. Hirose, Y. Haikawa, T. Takenaka, en Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '98), vol 2, pp: 1321-1326. 1998.

[2.49] "Framework of Distributed Simulation System for Multi-agent Environment", Itsuki, N. en RoboCup 2000: Robot Soccer World Cup IV, Springer, 2001, Peter Stone, Tucker Balch, Gerhard Kraetzschmar ed., pp.229-238. 2000.

[2.50] "Multilevel analysis of classical conditioning in a behaving real world artifact", P.F.M.J. Verschure, J. Wray, O. Sporns, G. Tononi, G.M Edelman, en Robotics and Autonomous Systems, 16, pp 247-265. 1996.

[2.51] “The RoboCup Physical Agent Challenge”, M. Asada, Y. Kuniyoshi et. al. en the XV Int. Conf. on Artificial Intelligence (IJCAI'97), pp. 51-56. 1997

[2.52] "Negotiating task decomposition and allocation using partial global planning", E. H. Durfee and V. Lesser, en Distributed Artificial Intelligence Volume II. Pitman Publishing., L. Gasser and M. Huhns, eds, pp: 229–244. 2009.

[2.53] “The RoboCup physical agent challenge: Goals and protocols for phase I”, Asada M., Stone P., Kitano H. et all en Lecture Notes in Computer Science, 1998. Ed. Springer Berlin / Heidelberg, Isbn: 978-3-540-64473-6, pp 42 a 61, volume 1.395. DOI: 10.1007/3-540-64473-3_48

[2.54] "Negotiating task decomposition and allocation using partial global planning", E. H. Durfee and V. Lesser, en Distributed Artificial Intelligence Volume II. Pitman Publishing., L. Gasser and M. Huhns, eds, pp: 229–244. 2009.

[2.55] "Readings in Distributed Artificial Intelligence", A. H. Bond and L. Gasser, eds., Morgan Kaufmann Publishers, 1988.

[2.56] "Social conceptions of knowledge and action", L. Gasser, en Artificial Intelligence, 43(1), 107-138, 1991.

[2.57] "DAI betwist and between: From intelligent agents to open systems science". C. Hewitt, J. Inman, en IEEE Trans. SMC, 21(6), 1409–1418, 1991.

[2.58] "Negotiation as ametaphor for distributed problem solving", R. Davis, R. G. Smith, en Artificial Intelligence, 20, 63-100, 1983.

[2.59] "An implementation of the contract net protocol based on marginal cost calculations", T. Sandholm en Proc. of the 11 Nat. Conf. on Artificial Intelligence, pp: 256–262. Washington, July 1993.

Capítulo 3. Arquitectura del Sistema

C.3.

3.1. Descripción general

3.1.1. Introducción

En este punto se describe el Sistema GdRBot, especialmente su componente de software. El sistema GdRBot pretende llegar a ser una referencia básica para el diseño de un sistema robótico, flexible y fácil de usar.

El sistema está formado por tres tipos de actores: clientes, servidores y elementos de red, figura 3.1. Estos actores pueden ser ejecutados en diferentes tipos de plataformas hardware.

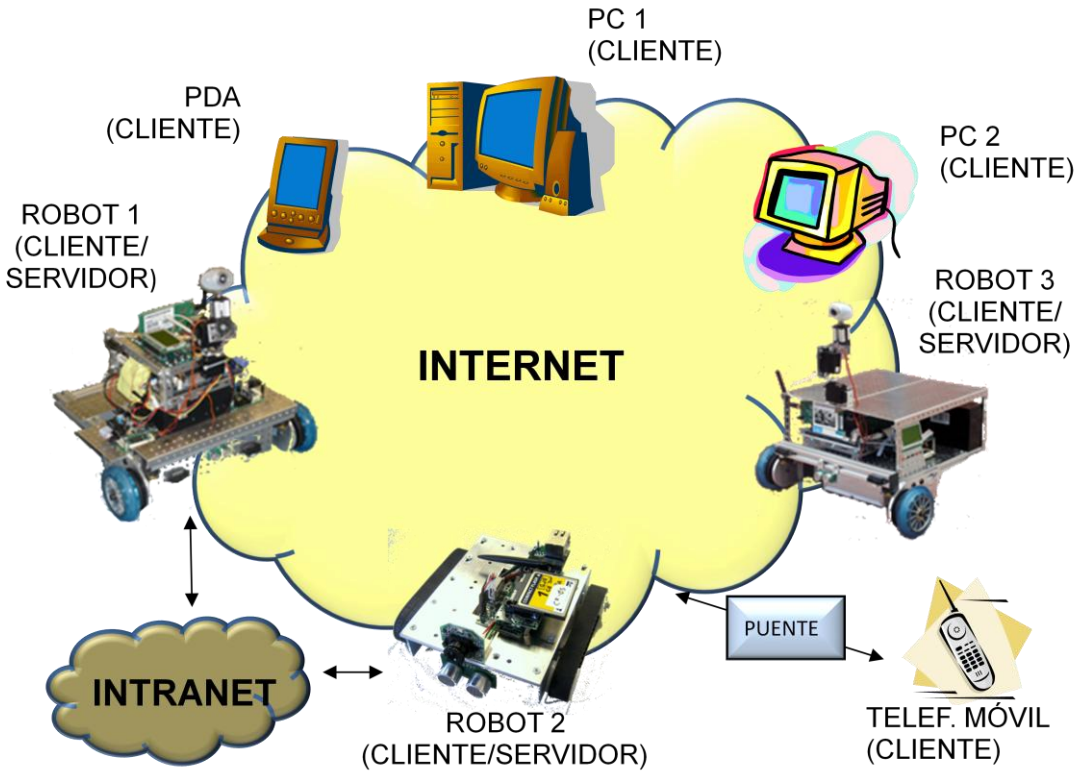


Figura 3.1. Arquitectura del sistema

El cliente realiza solicitudes y supervisa los servidores a través de llamadas XML-RPC [3.1]. El servidor y el cliente no tienen porqué comunicarse a través del mismo medio, ya que los elementos de red pueden unir distintos segmentos de red de forma

transparente. El servidor está instalado en los robots y los clientes pueden instalarse en el propio robot o en cualquier ordenador, *smartphone, tablet,* etc.... Los clientes instalados dentro de los robots se utilizan para realizar tareas de supervivencia, como evitar accidentes y colisiones, y hacer tareas de cooperación, como jugar al fútbol.

La figura 3.1 muestra la arquitectura propuesta para el sistema con todos sus elementos conectados a través de una red.

En resumen, el servidor es el encargado de realizar la gestión sobre el hardware y el cliente hace uso de ese hardware para lograr una tarea.

En la práctica, cada robot estará construido en base a un elemento de control relativamente potente, con una o varias interfaces de red y un número no definido de sensores/actuadores, que dependerán de la aplicación final para la que esté destinado, de forma que pueda ser útil en el mayor número posible de escenarios. La plataforma debe manejar todos estos elementos de una forma sencilla y flexible.

3.1.2. Descripción de la Arquitectura

El sistema debe soportar tres tipos de acciones:

Supervivencia: el robot debe ser una entidad autónoma, con la posibilidad de tomar decisiones por sí mismo. Por ejemplo, evitar accidentes.

Control remoto: el usuario debe ser capaz de controlar un robot de forma remota para llevar a cabo acciones, como el control de movimiento, o para mostrarnos sus sensaciones del entorno donde se está moviendo, como puede ser, lo que esté captando en cada instante con una cámara de vídeo.

Interoperabilidad: los robots deben ser capaces de interactuar entre ellos para llevar a cabo tareas comunes, como por ejemplo el diseño del plano de una planta o jugar un partido de fútbol.

Para proporcionar estas acciones se proponen dos opciones:

i) Un servidor para cada tipo de acción en cada elemento de control del robot.

ii) Un único servidor para llevar a cabo las tareas de cada elemento de control del robot.

Si se opta por la primera solución, en la que se van a tener varios servidores, se debe tener en cuenta que hay que enfrentarse a problemas como el arbitraje de los accesos a los actuadores/sensores del robot. Por ejemplo, cuando se quiera avanzar hacia un punto de forma remota y el sistema de supervivencia opta por lo contrario. La solución para este problema pasa por la implementación de árbitros que garanticen el acceso equitativo y ordenado al hardware del robot. Esta solución complica el desarrollo del control del hardware. Se deberán generar elementos diferentes, con diferentes interfaces, multiplicando los esfuerzos necesarios para la resolución de las mismas tareas. Sin embargo, como un aspecto positivo, se dispondrá de un software especializado y muy eficiente que admitirá el control de los diferentes dispositivos a través de la gestión de diferentes interrupciones.

Si la solución elegida es la segunda, con un solo servidor, no aparecerán problemas de competencia por el hardware, ni habrá que duplicar esfuerzos en el desarrollo de los drivers correspondientes. Sin embargo, el software será menos eficiente al no utilizar la gestión a través de interrupciones. De todas formas, se puede mejorar el rendimiento utilizando mecanismos de agrupación de tareas. En cualquier caso, el servidor debe tener consciencia de todos los sensores del robot.

Dado que la simplicidad es una de las claves en la plataforma propuesta, se ha optado por un sistema construido por un único servidor.

3.1.3. Transporte

Para el transporte de los mensajes entre los elementos del sistema, es decir en la comunicación entre los robots se ha elegido TCP/IP.

Los protocolos basados en TCP/IP admiten multitud de soportes físicos. Esto es importante pues en robótica hay algunos escenarios donde el uso de un soporte físico particular está totalmente descartado y otros donde las condiciones de trabajo son tan específicas que sólo se puede utilizar uno muy determinado, como puede ser por ejemplo en aplicaciones espaciales.

En la actualidad se pueden ver implementaciones de TCP/IP sobre Ethernet, 802.11, puerto serie, puerto paralelo, módem GSM, USB, Bluetooth e IrDa, radio enlace, etc...

Así mismo, si la seguridad es un requisito, o se quieren interconectar redes en distintos lugares del planeta, se pueden usar redes privadas virtuales.

3.1.4. Procesamiento remoto

Una vez que se ha decidido una arquitectura basada en un único servidor para cada robot, hay que decidir la manera de comunicarse con este servidor. Se necesita un sistema simple. Si el desarrollo se complica la plataforma pasa a no tener sentido, y debe soportar de manera sencilla la mayor cantidad de lenguajes de programación diferentes, de forma que no se debe forzar al equipo de desarrollo que quiera utilizar la plataforma a utilizar un lenguaje en particular. Por último se deberá elegir un protocolo estándar, ya que se utilizará desde diferentes sistemas operativos.

Las alternativas más maduras que existen en la actualidad son: CORBA, DCOM, SOAP, RMI y XML-RPC. A continuación se realiza una breve descripción de cada una de ellas:

- CORBA (*Common Object Request Broker Architecture*): es un lenguaje popular para escribir software distribuido orientado a objetos. Es muy compatible y posee una descripción de interfaz (IDL) muy eficaz, pero es muy complejo. Requiere que los clientes sean muy sofisticados y es difícil de aplicar [3.2].
- DCOM (*Distributed Component Objetc Model*): es la respuesta de Microsoft a CORBA. Es más fácil de utilizar; sin embargo, sólo se encuentran implementaciones maduras para Microsoft Windows [3.3].
- SOAP (*Simple Object Access Protocol*): está basado en XML+HTTP, pero la especificación no es muy buena y tiene elementos innecesarios [3.4].
- RMI (*Java Remote Method Invocation*): es el sistema de intercomunicación de Java. Es muy potente y fácil de usar, sin embargo, sólo se puede utilizar desde Java. No es una buena opción para el sistema que se pretende desarrollar ya que se busca todo lo contrario, que sea lo más genérico y que tenga el mayor soporte posible [3.5].
- XML-RPC: se basa en el protocolo HTTP para transporte y en XML para el código. Por lo tanto, es un sistema altamente estandarizado y fácil de utilizar. Asimismo, posee implementaciones para casi cualquier lenguaje conocido. En

el caso de utilizar un lenguaje que no lo soporte, es fácil de añadir la implementación de XML-RPC, su estándar no tiene más de 10 páginas. En resumen, XML-RPC implica sencillez y estándar. Como punto negativo es necesario poner de relieve la sobrecarga causada por HTTP y XML. En la actualidad, hay más de 100 implementaciones oficiales de XML-RPC que dan soporte a más de 40 lenguajes de programación [3.1]. Otra gran ventaja de XML-RPC es que soporta Reflexión, es decir, los servicios pueden ser descritos por ellos mismos. Es decir, para insertar un nuevo servicio no hace falta modificar el sistema, él solo aporta todo lo necesario para su integración con el resto. Para finalizar, resaltar la última gran ventaja de XML-RPC: los robots están abiertos para el mundo de la Web. Hoy, el protocolo que utilizan los sistemas web para intercomunicar suele ser XML-RPC. Usando XML-RPC se está conectando el robot al exterior, a la nube, etc...

Con todo lo descrito en los puntos anteriores, la elección final ha sido XML-RPC.

3.1.5. Sistema operativo del servidor

Una vez que se ha seleccionado el protocolo de alto nivel, el procedimiento para llamadas remotas y el diseño del servidor, el siguiente paso es la selección del sistema operativo que debe residir en el servidor.

Esta elección es una de las más importantes del sistema, porque va a afectar significativamente a la eficiencia, la adaptabilidad y la flexibilidad de la plataforma.

Ya que la comunicación se hará a través de XML-RPC + TCP/IP, la interoperabilidad está garantizada, pero se tendrán que analizar otros elementos diferentes a éstos, como rendimiento, conectividad y compatibilidad con diferentes arquitecturas hardware

En este caso, y tras una serie de análisis, la opción más indicada es Linux. Linux es fácilmente modificable por el usuario final, ya que dispone de una gran cantidad de documentación y fuentes a disposición del usuario, del mismo modo es compatible con una gran cantidad de arquitecturas (32-bit, Compaq Alpha AXP, Sun SPARC y UltraSPARC, Motorola 68000, PowerPC, PowerPC64, ARM, Hitachi SuperH, IBM S/390, MIPS, HP PA-RISC, Intel IA-64, VAX de DEC, AMD x 86-64, AXIS CRIS y arquitecturas de

Renesas M32R). Esto permitirá seleccionar cualquier tipo de hardware e incluso sustituirlo durante la vida útil del sistema.

Por otro lado, versiones reducidas de Linux funcionan muy bien en plataformas hardware con pocos recursos, lo que puede permitir reducir costes en ciertas aplicaciones que no requieran de mucha potencia de cálculo.

En contra, cabe resaltar que algunos fabricantes ocultan información sobre el uso de sus dispositivos, lo que dificulta su uso en Linux si dicho fabricante no proporciona los drivers correspondientes. Sin embargo, se puede afirmar que la elección de Linux será la opción que permitirá la inclusión de una mayor variedad de elementos hardware, entre todas las opciones posibles.

Para finalizar, otra razón para la elección de Linux como sistema operativo es que está enfocado a la red y a la seguridad. El núcleo de Linux tiene soporte para infinidad de medios físicos para comunicaciones IP, así como de enrutamiento, de puente, de firewall, etc.…

En cualquier modo, puesto que la base del sistema es XMLRPC y TCP/IP, podemos afirmar que es muy sencillo portarlo a cualquier otro sistema operativo, ya sea Windows, MAC o incluso Android.

3.1.6. Lenguaje de programación del servidor

La última decisión importante que queda por tomar es el lenguaje de programación a utilizar para implementar el servidor de robot.

Para la toma de esta decisión, hay que tener en mente que el protocolo XML-RPC utiliza HTTP como protocolo de transporte, por lo que también se necesitará un servidor Web. Si se dispone de un robot con suficiente capacidad se puede utilizar Apache como servidor Web, pues ya lo incluye Linux. En el caso de que la CPU no tenga suficiente capacidad de cómputo se podrá optar por otros servidores más ligeros o incluso implementar uno a medida.

Una vez resuelto el problema del protocolo HTTP, hay que elegir un lenguaje de programación suficientemente testado, estable y potente que permita una comunicación con el hardware de una forma sencilla y directa. Para ello, puede

parecer que el uso de un lenguaje interpretado es la solución, ya que permitiría embeberse en el sistema operativo, pero más adelante, en algunas pruebas realizadas, se verá que esta solución no sólo hace que se aleje del sistema operativo y, por lo tanto, también lejos del hardware, sino que también reduce el rendimiento. Dado que uno de los objetivos del servidor es exactamente lo contrario, facilitar el acceso al hardware y su cercanía, el lenguaje no debe ser compilado.

Por lo descrito en el párrafo anterior, se puede interpretar que la mejor opción es el uso de un lenguaje que está muy relacionado con el hardware, tal como el ensamblador. Sin embargo, hay que tener en cuenta que también hay que llevar a cabo tareas complejas desde XML-RPC, lo que sería muy costoso hacer desde un lenguaje de tan bajo nivel. Además, sería una gran restricción del sistema pues estaría completamente unida a una plataforma de hardware determinada.

Por todo esto, se concluye que el punto de inflexión entre un lenguaje de alto nivel, como puede ser Java, y un lenguaje de bajo nivel como ensamblador, es el lenguaje C. Este lenguaje permite realizar tareas de muy alto nivel, debido a la gran cantidad de bibliotecas disponibles, mientras que a la vez no se aleja de los equipos, para que se pueda continuar el acceso a bajo nivel de una manera sencilla

En resumen, la plataforma software propuesta, está formada por dos tipos de elementos: servidores y clientes. Hay un servidor en cada robot atendiendo las peticiones de los clientes que pueden residir en cualquier sistema hardware dotado de un procesador, incluidos los propios robots (en este caso para realizar tareas de supervivencia o de cooperación con otros robots). La arquitectura permite realizar acciones de tres tipos: supervivencia, control remoto e interoperabilidad. Para la comunicación entre clientes y servidores se utiliza XML-RPC sobre HTTP a nivel de aplicación (mediante un servidor web Apache), TCP/IP para el transporte y numerosas posibilidades de capa física (USB, Bluetooth, puerto serie, puerto paralelo, GSM/GPRS, radio y WLAN). El sistema operativo para los servidores es Linux. Internamente los servicios están programados en lenguaje C. El cliente es una aplicación programada en cualquier lenguaje para el que se disponga de una librería robótica encargada del intercambio de mensajes XML-RPC, con independencia del sistema operativo [3.6]

3.2. Diseño de la plataforma

Una vez analizadas las necesidades de desarrollo, se describe el diseño de la plataforma, basado en una estructura de tipo cliente-servidor.

3.2.1. El cliente

El cliente es un programa o servicio que realiza las peticiones a un servidor, o lo monitoriza. Se compone de tres partes, mostradas en la figura 3.2:

- **Aplicación:** se diseña por el usuario final y se desarrolla en el lenguaje seleccionado por él. Debería ser diseñado para que sea capaz de ejecutarse en cualquier sistema operativo.
- **Librería de Robótica:** se suministra con la plataforma desarrollada en este trabajo y está disponible en varios lenguajes de programación. Simplemente abstrae al usuario del protocolo XML-RPC.
- **Librería de XML-RPC:** es suministrada por una tercera parte. Implementa el protocolo XML-RPC. Esta librería se basa en estándares abiertos, por lo que si fuera necesario se podría desarrollar desde cero, sin ser demasiado complejo.

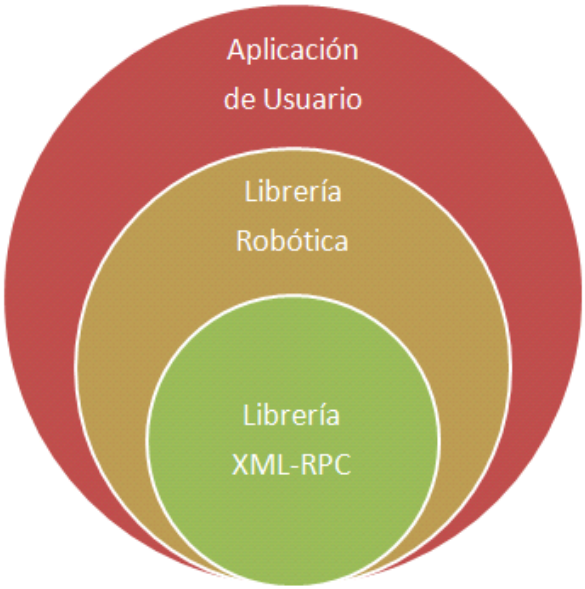


Figura 3.2. Estructura del Cliente

3.2.2. El servidor

El servidor proporciona acceso a los sensores y actuadores a través de la interfaz XML-RPC [3.1] y está instalado en el robot. Como sensores se han conectado detectores de distancia por infrarrojos, codificadores de posición ópticos, interruptores de final de carrera, cámaras de vídeo e incluso otro tipo de sensores controlados a su vez a través de un microcontrolador dedicado. Como actuadores se han usado motores de corriente continua, motores paso a paso y desplazadores lineales. El servidor también puede operar con otros servidores para proporcionar una información más compleja que pueda necesitar el cliente.

Básicamente, se compone de las siguientes partes (figura 3.3):

- Interfaces de red: Permiten al robot la comunicación entre diferentes redes y distintos medios físicos. Se puede comunicar con el cliente o con otros servidores. La única restricción que debe cumplir la interfaz es disponer de una pila TCP/IP en la capa superior. En el presente trabajo se han probado, con éxito, las siguientes interfaces:
    - Ethernet
    - Wi-Fi (802.11)
    - Bluetooth
    - GPRS
    - Puerto paralelo
    - Puerto serie asíncrono RS-232
    - Puerto USB
- Servidor Robótico: Lleva a cabo las acciones lógicas del servidor. Recibe peticiones XML-RPC a través de las interfaces de red, que deben ser tratadas de forma independiente. Puede comunicarse con el hardware a través de diferentes elementos, tales como puertos serie (RS-232, I2C, SPI), PCI, puerto paralelo, USB, etc. Desde un punto de vista global, el Servidor Robótico se compone de:
    - Servidor WEB: Su función es la de negociar y tramitar las solicitudes HTTP usando CGIs, transmitiendo estas peticiones al servidor del sensor.
    - Unidad de Control: Se ejecuta a través de CGI y su función es la de analizar las solicitudes del servidor Web. Procesa esa solicitud y

devuelve una página Web donde se codifica la respuesta. A su vez, el servidor de sensores se compone de una librería XML-RPC. Analiza las solicitudes realizadas por el servidor Web y evalúa sus parámetros. Del mismo modo, se ejecuta la función correspondiente para esta solicitud. Internamente está compuesta por dos librerías: HTTP y XML. Se incluye un *Dispatcher* que arbitra el acceso a los diferentes dispositivos para evitar conflictos. Asimismo, distribuye las diferentes solicitudes a los diferentes *drivers*.

- Servidor Sensorial. Gestor de comandos que permite la comunicación entre el servidor robótico y el hardware a controlar, como sensores y actuadores.
    - Núcleo. Permite el acceso al hardware (tanto a dispositivos de red como a los diferentes sensores / actuadores).
    - Drivers. Gestionan el acceso a bajo nivel a los diferentes dispositivos hardware.

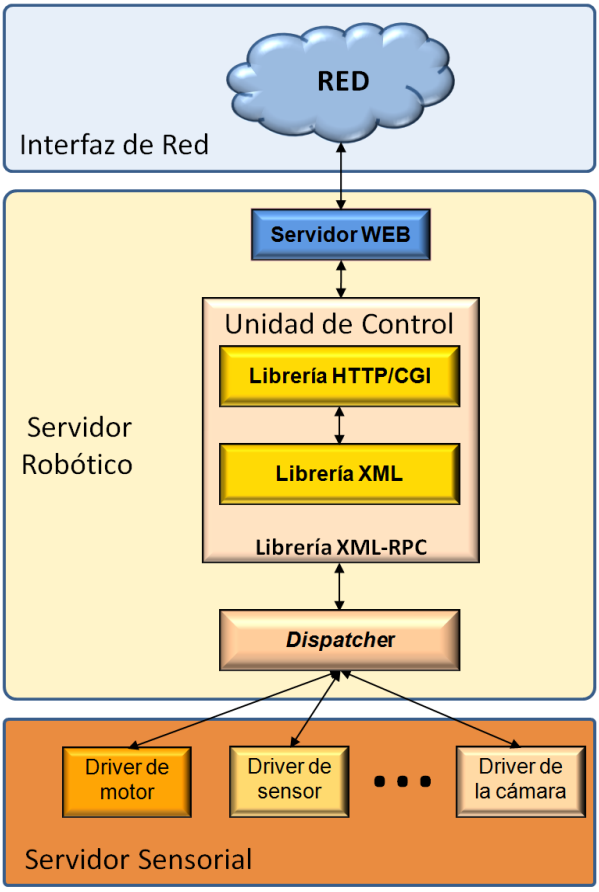


Figura 3.3. Diagrama de bloques del servidor

3.3. Descripción de la interacción Cliente/Servidor

Una vez detallado el diseño de la arquitectura a alto nivel, a continuación se va a describir cómo se gestiona una solicitud por los diferentes elementos de la plataforma (figura 3.4):

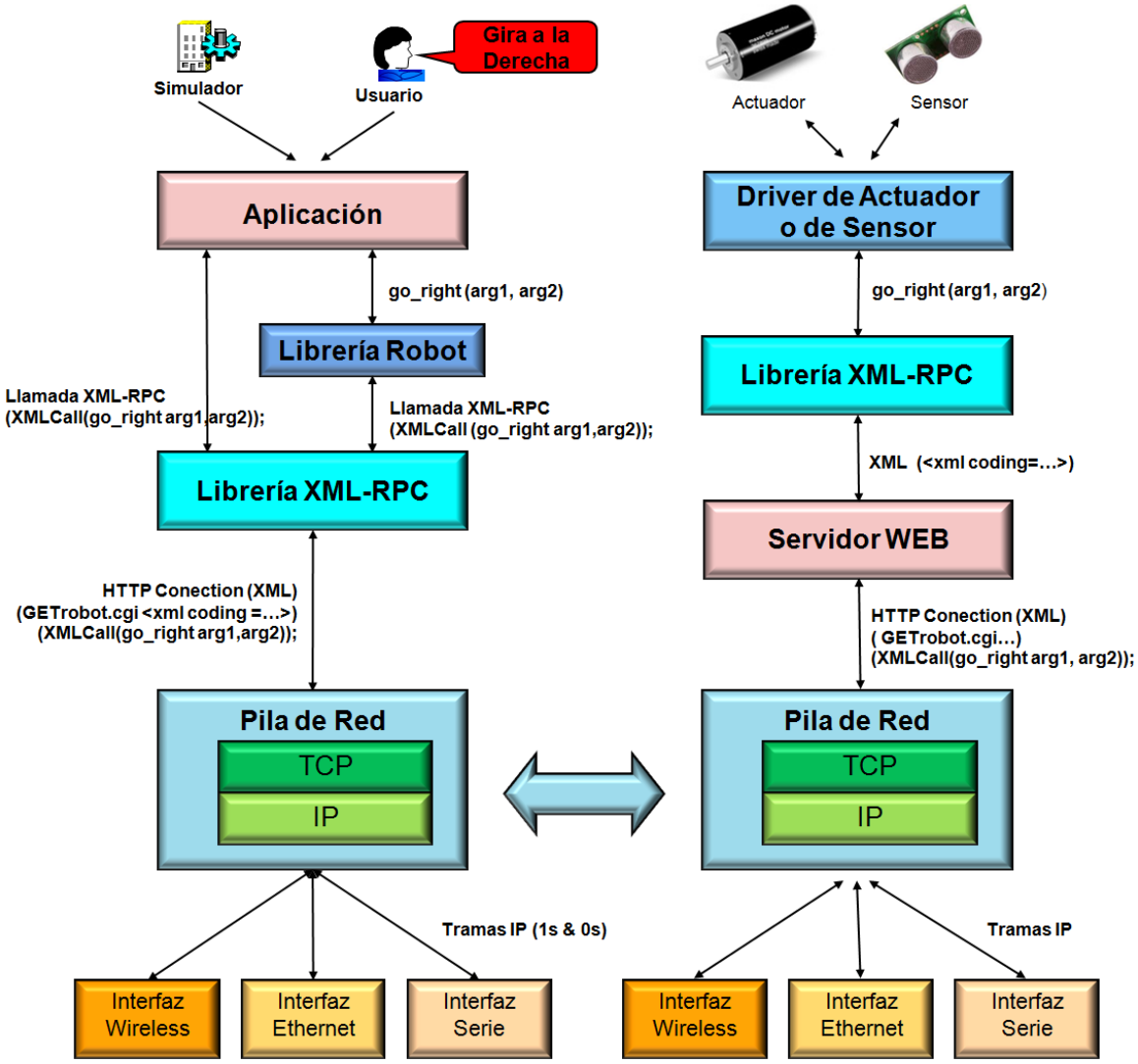


Figura 3.4. Arquitectura de la pila de comunicaciones.

a) Una aplicación de usuario decide realizar una petición, por ejemplo girar a la derecha, para lo cual deberá utilizar la librería robótica o la librería XML-RPC. Si la solicitud es incorrecta o no cumple con la sintaxis correcta, se devuelve un error.

b) Una vez comprobado que es correcta, la solicitud se transforma en un documento XML que el cliente lo transmitirá a través del protocolo HTTP.

c) La comunicación se divide en tramas TCP/IP que viajarán a través de una de las diferentes interfaces de red en función del destino del mensaje. En el caso de que se pierda parte del mensaje, TCP las retransmite de forma que exista una completa transparencia semántica.

d) Si es necesario, un elemento de red retransmitirá la petición a través de diferentes redes físicas, por supuesto sin modificar su contenido.

e) Una vez que la trama ha llegado a su destino, en el servidor, los paquetes se decodifican para formar una petición HTTP que será transmitida al servidor WEB.

f) El servidor Web identifica el mensaje y lo envía al servidor de robótica utilizando CGI. Si no se completara correctamente el protocolo HTTP se genera una página de error.

g) El servidor robótico obtiene el documento XML que será analizado por la librería de XML-RPC. Si se solicita un comando no implementado o el protocolo no se ha completado, se devuelve un documento XML con un mensaje de error.

h) Los drivers pasan la solicitud a los diferentes elementos hardware.

i) La respuesta obtenida de los drivers se transforma en un documento XML utilizando la librería XML-RPC.

j) El documento XML se transmite al servidor Web que lo retransmite al cliente, realizando todos los pasos anteriores en orden inverso.

3.4. Descripción modular

3.4.1. Nivel Físico-Sensorial

Este nivel se ocupa de estimular convenientemente a los sensores y actuadores. Su misión es abstraer al usuario de la complejidad electrónica y a veces de la temporización crítica que requieren algunos sensores, de manera que la capa del nivel superior sólo perciba acciones y observaciones completas. Su concepción permite la realización de acciones concurrentes, por ejemplo mientras se lleva a cabo la acción de avanzar las ruedas motrices cincuenta vueltas y media, se pueden medir distancias y mover servomotores.

Sobre el hardware que soporta este nivel se ejecuta un software que actúa como un servidor de servicios básicos. La dinámica de funcionamiento consiste en recibir órdenes o comandos a través de un puerto de comunicaciones (ya sea vía radio o por cable), y responder por el mismo puerto con los valores solicitados de los sensores, códigos de aceptación o códigos de error. De esta manera al arrancar el sistema la capa superior puede escanear los servicios disponibles evaluando los retornos a los servicios solicitados. Esto permite añadir tantos sensores o actuadores como sean necesarios sin necesidad de modificar el software que se ejecuta en la unidad central, cada nuevo servidor se identifica e indica el tipo de servicio que aporta al sistema.

A esta estructura hardware y software se la ha denominado **Servidor Sensorial**. Los Servicios del Servidor Sensorial son aquellos servicios gestionados por el microcontrolador. A continuación se describen algunos de los "Servicios del Servidor Sensorial" creados para algunos de los Servidores Sensoriales que se han desarrollado en esta tesis:

Servicios de Identificación

Solicitud y asignación de un identificador a cada robot, la respuesta se realiza con un retardo aleatorio para evitar identificadores repetidos. Se precisa una confirmación parte del PC. Se realiza un *Ping* a cada robot, enviando el comando “P”.

Trama envío: 1 byte : **0x50** (ascii de 'P') $\rightarrow$ (solicitud de *ping*)

Trama respuesta: 1 byte : **0x4F** (ascii de 'O') $\rightarrow$ (respuesta Ok)

Servicios de Motricidad

Encoder: Permite la lectura y la actualización de los contadores de transiciones en los encoders para control de motores de continua. Esta orden precede a un movimiento indefinido para limitar el giro. Comando "E".

Trama envío: **0x45** $\rightarrow$ (comando “Encoder”)

Trama respuesta: **pasos0, pasos1, pasos2, pasos3** $\rightarrow$ (número de transiciones pendientes de cada motor)

Trama envío actualización: **p0, p1, p2, p3** $\rightarrow$ (valores nuevos)

Motores: Comando que gestiona las órdenes para realizar movimientos indefinidos.

Trama envío: **0x30..0x39** → ('0'..'9', ver teclado numérico en figura 3.6)

Trama repuesta : **-** → (no hay respuesta)

Servomotores: Utilizado para el posicionamiento de los servomotores. Comando "W".

Trama: **0x57**, **#serv(0..n-1)**, **pos (0x2d..0xae) ~0..180 grados** → (comando "W", el número de servo que se quiere modificar, por si hay más de uno, y la posición seleccionada)

Servo 0: PTD0, Servo 1: PTD1 → (esta es la configuración utilizada con el Servidor Sensorial GP_Bot, descrito en el Anexo I. Se han conectado dos servomotores, uno en el puerto PTD0 y otro en el PTD1)

Este comando puede ser muy dependiente del tipo de servomotor utilizado, por lo que puede ser necesario realizar una calibración de los parámetros de posicionamiento para cada modelo de servo.

Servicios de Percepción

Conversor analógico-digital: utilizado por ejemplo para leer distancias con sensores de infrarrojo analógicos. Comando "A"

Trama de envío: **0x41**, **#canal (0..n; GPbot: B0..B7)** → (comando "A" y selección del canal que se quiere leer)

Trama de respuesta : 0..255 → (devuelve la lectura del conversor, en 8 dígitos)

Estado puertos: obtiene de forma rápida todas las entradas digitales del sistema. Utilizado por ejemplo con el uso de lector de distancias por infrarrojos Digital. Comando "D".

Trama envío : **0x44** → (comando "D")

Trama recepción: **Puerto A, Puerto B, Puerto C, Puerto D**. (4 bytes) → (devuelve 4 bytes, uno con el contenido de cada puerto. En el caso de la GP_Bot son del A al D.

Servicios de Comunicación

Mensajes: Se envían los comandos y los parámetros necesarios para la comunicación entre la unidad central y el Servidor Sensorial.

Trama de envío: **45 7D 25 32 45** → ("E", "ID_remitente", "A", "B", "E")

Trama de recepción: Como todos reciben el mensaje, todos no pueden confirmar la recepción. Serán los comandos particulares los que originen mensajes de respuesta, es decir, sólo responderá el destinatario.

Servicios Genéricos

Store: Almacena un valor en una dirección de memoria determinada. El Servidor Sensorial dispone de una pequeña memoria donde poder almacenar valores. Se utiliza, por ejemplo, para guardar valores de parámetros que se pueden ir modificando a lo largo de la ejecución de la aplicación. Comando “S”.

Trama envío: **0x53**, **ByteDirL, ByteDirH, ByteNuevoValor** → (comando "S", la dirección de escritura (2 bytes) y el valor a escribir (1 byte)).

Trama respuesta: **0x53** → (comando "S").

Load: Obtiene el valor de un dirección determinada. Es el comando complementario al anterior. Comando "L".

Trama envío: **0x4C**, **ByteDirL, ByteDirH** → (comando "L" y la dirección de lectura (2 bytes))

Trama respuesta: **0x4C, ByteValor** → (comando "L" y el valor leído (1 byte))

Aunque, sin lugar a dudas, el servicio más importante de todo Servidor Sensorial es aquel que contiene todas las funciones o servicios que puede aportar, con el detalle de cómo se usan, como funcionan y para qué sirven. De esta manera, si se conecta un Servidor Sensorial nuevo a un sistema, éste lo identifica como nuevo y lo primero que hace es preguntarle ¿tú que sabes hacer?. El Servidor Sensorial le contesta con la lista de servicios que aporta y la forma de acceder a ellos, con la descripción de todas sus funciones accesibles y los parámetros involucrados, tanto de entrada como de salida. Toda esta información le llega al cliente y, a partir de ahí, podrá utilizar todas las nuevas funcionalidades.

3.4.2. Nivel Control-Aplicacional

Esta capa proporciona una API en C para la implementación de arquitecturas multi-agente escalables. Estos servicios permiten la utilización de los sensores y actuadores de manera rápida y cómoda. Su misión es proporcionar rapidez de desarrollo y potencia de cálculo. Se pueden configurar varias posibilidades:

Una unidad de control (PC o similar) por cada agente móvil (figura 3.5): Si se opta por un PC dedicado a cada robot móvil, lo más lógico es que éste vaya integrado a la estructura móvil y conectado directamente al Servidor Sensorial.

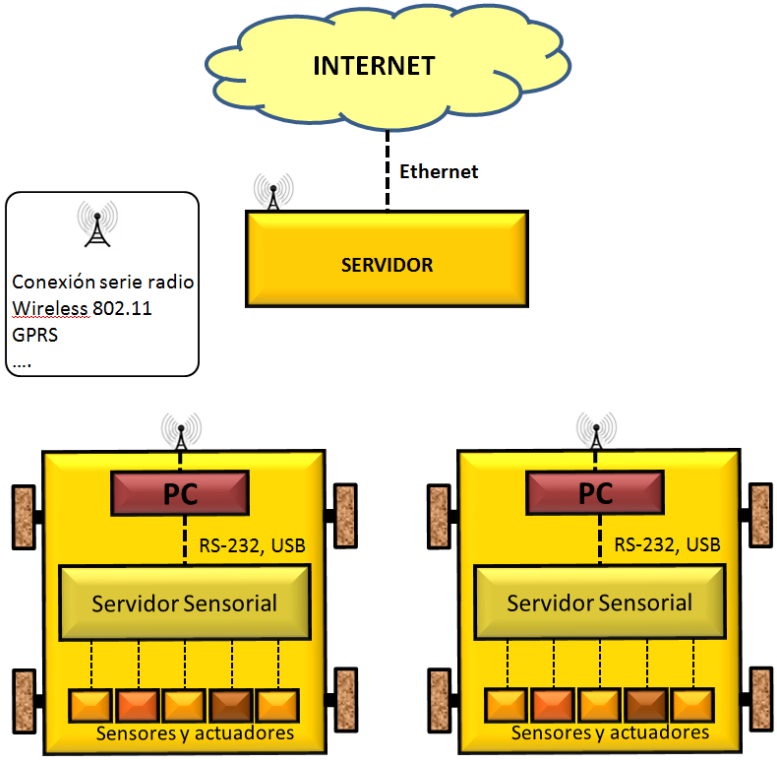


Figura 3.5. Los robots se comunican con el servidor a través de distintos PCs

Un PC para varios agentes móviles (figura 3.6): Si se comparte un PC para varios robots estos se comunicarán con él vía radio a través del mismo puerto. En el PC se ejecutará un hilo cliente por cada robot, no compartiendo información dentro del PC (para mantener autonomía). Al arrancar el programa cliente el sistema asigna un identificador (p.e. el primer robot que responda a un ping). El cliente una vez asignado su robot manda tramas precedidas por un identificador que identifica al destinatario, de manera que solo su robot se dé por aludido. Aparte del algoritmo personalizado por la aplicación concreta, el cliente proporcionará controles básicos de movimiento del robot, (p.e. para evitar que se estrelle, reset..). Si este PC está conectado a Internet, mediante sesión remota se podría controlar cada robot desde la propia Internet.

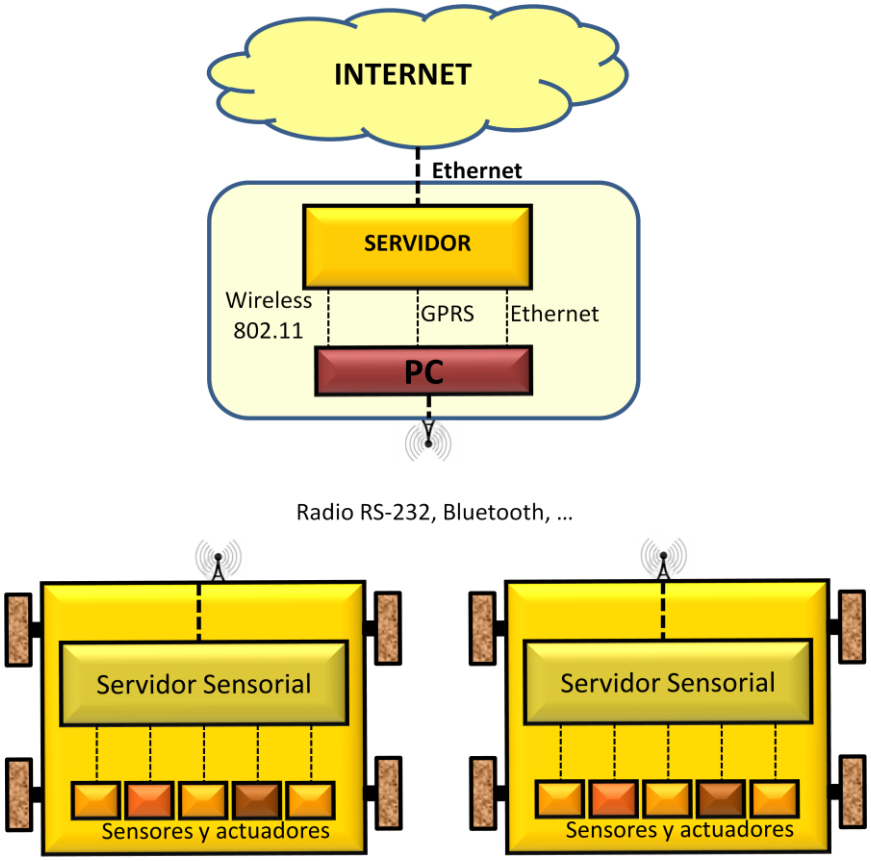


Figura 3.6. Los robots se comunican con distintos procesos de un mismo PC.

3.4.3. API de usuario en C

Aunque el API mantiene cierta independencia con la capa inferior en lo que se refiere a su utilización, guarda una dependencia total de los recursos según el número de dispositivos concretos conectados.

Para el conjunto de servicios se ha establecido un nivel de prioridad de desarrollo según se consideren servicios básicos para la experimentación de sistemas multi-agente. En este sentido se han de cubrir los campos de **comunicación, motricidad y percepción del entorno**. Para ello, respectivamente, se prioriza la disponibilidad de servicios de utilización de la radio, control de motores y sensores.

A continuación se muestran una serie de ejemplos de servicios de este nivel, realizados para el Servidor Sensorial GP_Bot, descrito en al Anexo I:

Servicios de lectura conversor analógico-digital:

Función: **int LeerAD_gpbot(int canal)** → (prototipo de función que realiza la lectura de uno de los canales del conversor analógico-digital. El parámetro `int canal : 0..7` se corresponde con el canal que se quiere leer, que en el caso de la GP_Bot se corresponde con uno de los pines del Puerto B, del PTB0 al PTB7 respectivamente)

Salida: **0..255** → (valor analógico leído, en formato digital, de 8 bits) ó
-1 → (si ha habido algún error)

En caso de utilizar hardware distinto en la capa inferior se utilizaría la misma nomenclatura de canales (0..n) según el número de canales/pines de que se disponga.

Servicios de infrarrojos:

Función: **int LeerIRanalog_gpbot(int id)** → (lectura del sensor analógico de infrarrojos indicado, en donde int id es el número del sensor a leer (0..3 por defecto GP_Ifaz))

Salida: **0..255** → (valor analógico leído, en formato digital, de 8 bits) ó
-1 → (si ha habido algún error)

Función: **int LeerIRdig_gpbot(int id)** → (lectura del sensor digital de infrarrojos indicado, en donde int id es el número del sensor a leer (0..3 por defecto GP_Ifaz))

Salida: **0/1** → (la respuesta es la lectura digital del sensor) ó
-1 → (si ha habido error)

Servicios de motores:

Función: **Movimiento_gpbot(unsigned char*mov, unsigned char*pasos),** en donde:

unsigned char*mov : → 0x30..... → 0x39 (tipo de movimiento según el teclado numérico '1'..'9', se muestra en la figura 3.7)

unsigned char*pasos: → (número de pasos del *encoder* a mantener dicho movimiento)

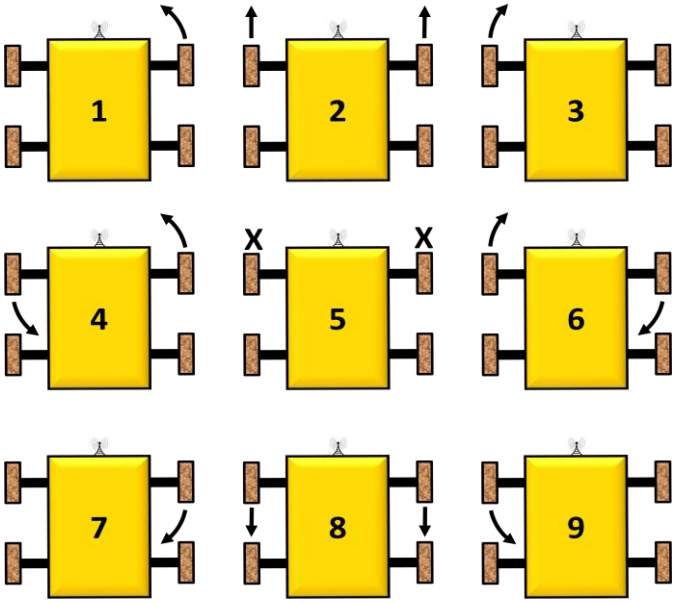


Figura 3.7. Tipo de movimiento según el teclado numérico.

En la figura 3.7, se muestra el movimiento de las dos ruedas motrices al activar cada una de las teclas del teclado numérico.

Según el diámetro de las ruedas y el número de pasos de avance se puede calcular el desplazamiento, que se calcularía a este nivel en un PC. Esto tiene una prioridad inferior. También se ha incluido la opción de almacenar las órdenes en el PC y monitorizar de forma temporizada con la función **int Pos(int id)** el incremento de pasos de cada motor y la cuenta restante. El resultado genera vectores de movimiento que sirven para construir mapas y volver a la situación inicial.

Servicios de servos:

Función: **int Servo(int id, int pos),** en donde:

int id : número del servo a mover (0..n)

int pos : //posición relativa al centro (izq todo:+100, dcha todo:-100 grados)

Servicios de *timer*

Función: **int LeerTimer(int id)**

Se utiliza para lectura de sensores de ultrasonidos.

Precaución: Los *timer* se utilizan para generar señales PWM que controlan la velocidad de motores de continua, por lo que hay que regular el uso reservado por otros servicios.

3.4.4. Nivel Conectividad-Monitorización

Esta capa representa la escalabilidad máxima de acceso y control del sistema implementado mayoritariamente en la capa inferior. Su misión es la de implementar servicios web para configuración/gestión del sistema multiagente. En este nivel sí se debe compartir información del conjunto de los agentes, de manera más o menos accesible por parte de estos según el algoritmo en cuestión.

3.5. Ejemplo de uso de una aplicación

Como ejemplo de la capacidad de realizar trabajo colaborativo por parte del sistema GdRBot, en la figura 3.8 se presenta un esquema para resolver el problema de llevar a cabo un trabajo coordinado entre un grupo de robots, supervisado por un PC central.

Para este caso, se instalarán dos clientes diferentes en la plataforma, situados en los robots:

- Cliente de supervivencia: Comprueba constantemente condiciones de error en el robot, como choques, obstáculos, etc.
- Cliente de colaboración: Gestiona las tareas de colaboración entre los robots.

También habrá un cliente instalado en el PC que se encargará de la supervisión de los robots. Los clientes instalados en el robot se pueden desarrollar en cualquier lenguaje de programación, bajo el mismo sistema operativo.

En cada uno de los robots se instalará un Servidor, que será el que acepte los comandos vía XML-RPC enviados por cada cliente.

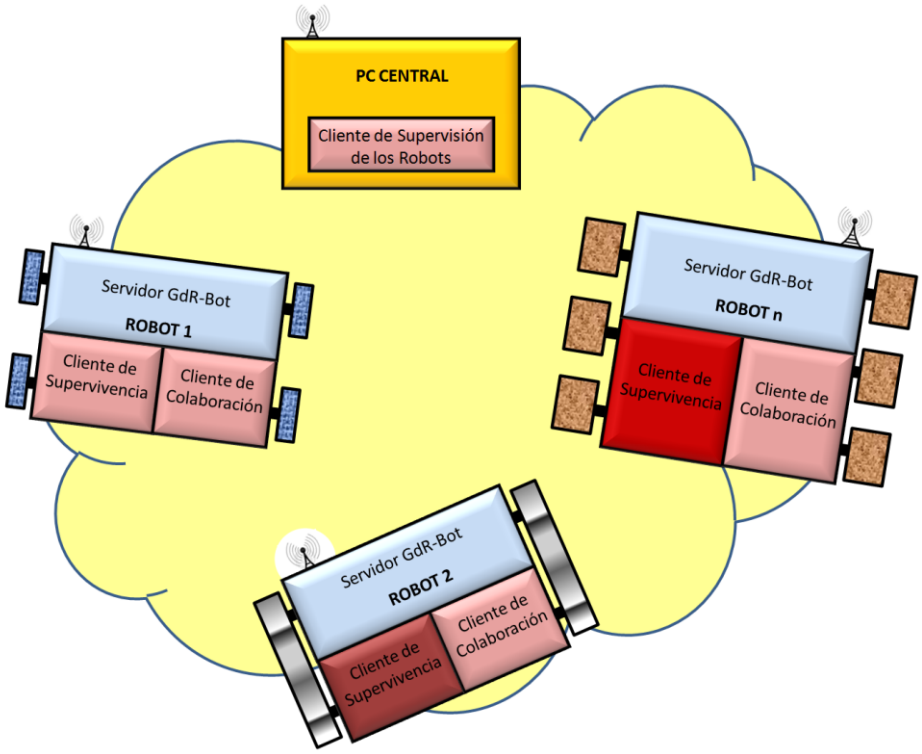


Figura 3.8. Esquema de una estructura Cliente-Servidor en GdRBot.

El cliente instalado en el PC se puede desarrollar en cualquier lenguaje de programación y en cualquier sistema operativo. Si el sistema operativo no es compatible con el medio físico de la red de robots, se utilizará un puente (*bridge*) que servirá de interconexión entre ellos.

3.6. Bibliografía del Capítulo 3

[3.1] Página web de XML-RPC, www.xmlrpc.com

[3.2] Página web de CORBA, www.omg.com/corba

[3.3] Página web de DCOM, www.opengroup.org/comsource

[3.4] Página web de SOAP, /www.w3.org/TR/soap

[3.5] Página web de RMI, www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

[3.6] "A Generic Software Platform for Controlling Collaborative Robotic System using XML-RPC", G. Glez. de Rivera, R. Ribalda, J. Colás, J. Garrido in Proc. IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics. (AIM'05), pp. 1336-41. Monterey (USA). July/2005.

Capítulo 4. Detalles de la Implementación

C.4.

4.1. Introducción

Como ya se ha comentado anteriormente, la plataforma GdRBot consiste en un conjunto ampliable de llamadas XML-RPC sobre una red heterogénea. Los robots se comportan como servidores, proporcionando acceso a sus sensores a través de una sencilla interfaz, accesible desde casi cualquier lenguaje de programación o sistema operativo. Los robots también pueden comportarse como clientes para acceder a otros robots, realizando tareas colaborativas y/o realizando intercambio de conocimiento. Todos los dispositivos de la arquitectura se pueden interconectar con cualquier interfaz: Ethernet, Wifi, Bluetooth, GPRS, USB,.etc....

La distribución de la arquitectura dependerá finalmente de la implementación concreta. En la figura 4.1 se muestran un esquema con todos los componentes que pueden intervenir y todas las comunicaciones posibles entre los módulos. Según la integración física de los módulos se optará por comunicaciones inalámbricas o cableadas. Así, la aplicación final impone qué módulos incluye la estructura móvil.

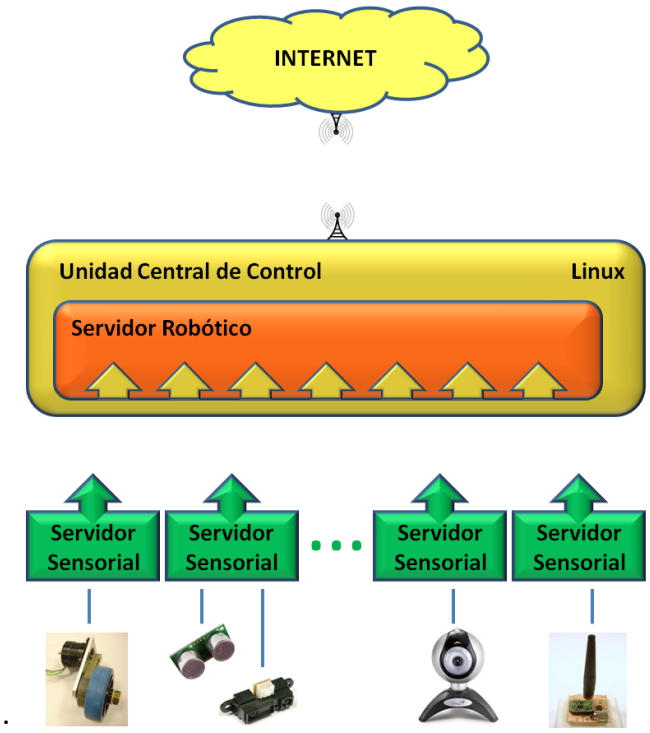


Figura 4.1. Diagrama de componentes de plataforma GdR_Bot

En el diseño concreto de un robot entran en juego diferentes factores que van a determinar los detalles de su construcción, tales como dimensión, velocidad máxima, fuerza, peso que pueda soportar, etc. A la hora de diseñar una plataforma genérica este punto cobra especial importancia puesto que inicialmente no se sabe el destino que va a tener, por lo que se intenta que sea lo más genérico posible pero a la vez suficientemente flexible para que se pueda adaptar a la mayor parte de las situaciones posibles.

Las características básicas de las que partimos pueden ser:

- Que sea robusto
- Flexible, de fácil ampliación o adaptación a diferentes entornos.
- Poco peso.
- Con materiales fáciles de encontrar en el mercado, que no sean propietarios.
- Bajo coste
- Con motores capaces de entregar un par elevado
- Que permita realizar movimientos con mucha precisión.

El primer prototipo y la parte principal del desarrollo se realizó utilizando un hardware basado en la placa base VIA EPIA TC 10.000. El desarrollo fue muy sencillo, pues básicamente es un PC, con lo que se disponía de múltiples herramientas de desarrollo y depuración, así como abundante documentación. Una vez terminado este primer desarrollo se comprobó de no era la mejor opción para muchas aplicaciones. El tamaño es relativamente grande, 17x17 cm, y presenta un gran consumo de energía, por lo que se debía acompañar de una batería de plomo. Todo el sistema resultó ser muy pesado, por lo que se necesitaron potentes motores para su tracción, lo que incrementó de nuevo tamaño y consumo.

Debido a que una de las características de la plataforma GdRBot es que no depende ni del software ni del hardware que lo soporta, sino que sólo depende de la especificación de un protocolo, puede ser implementada de una manera sencilla en cualquier dispositivo, cumpliendo algunos requisitos mínimos ya mencionados anteriormente.

Para probar lo anterior, y establecer diferentes comparativas, se han diseñado y construido otras implementaciones de la plataforma, basadas en hardware diferente.

Para realizar estas diferentes implementaciones hay que tener en cuenta una serie de aspectos software:

- ✓ Los dispositivos que van a ser los elementos de control necesitan, al menos, una interfaz de red y una pila TCP-IP para situar un servidor XML-RPC. En lugar de desarrollar una pila de red completa, se utilizará la del sistema operativo.
- ✓ Dicho sistema operativo también proporciona controladores para algunos sensores comunes y algunas aplicaciones como servidores web, que reduce drásticamente el esfuerzo de migración de la plataforma. Debido a que el primer robot que se diseñó con esta plataforma utiliza Linux, que puede ejecutarse en casi cualquier tipo de arquitectura, éste ha sido elegido como el sistema operativo de destino en el resto de las plataformas que se han implementado.

A continuación se van a analizar algunos aspectos en particular:

Compilador y LibC. Como se tiene previsto migrar los programas ya hechos a una arquitectura diferente a la máquina original será necesario el uso de un compilador. Un compilador cruzado permite compilar aplicaciones para otras arquitecturas diferentes de la que se está ejecutando dicho compilador. GNU GCC ofrece este servicio. Además del compilador, también se necesita la librería LibC para tener acceso a toda la funcionalidad del sistema desde las aplicaciones [4.1]. Ambos, GCC y LibC, se pueden obtener fácilmente de su página web y compilarlos automáticamente en la máquina de destino utilizando la herramienta Crosstool [4.2].

Kernel. Una vez conseguido el compilador cruzado, el siguiente paso es la obtención de un núcleo que se encargará de todo el control del hardware de nuestro robot, y proporcionará una interfaz de red al mismo. El código fuente del kernel se puede obtener de su sitio web oficial [4.3], siendo el mismo para cada arquitectura. Una vez descargado, se debe configurar para que se ajuste lo más posible al hardware final. Para esa configuración, se debe reunir toda la información posible sobre todos los elementos que componen la plataforma

hardware. Una vez configurado, se compilará el kernel, usando un compilador cruzado para el formato de imagen que necesita el cargador de arranque (*Boot Loader*).

Cargador de arranque (*Boot Loader*). El kernel es sólo un programa para el procesador del sistema de control seleccionado, se debe cargar en la memoria y el microprocesador debe comenzar a ejecutarlo desde su punto de entrada. Cada arquitectura tiene sus propias peculiaridades en el proceso de arranque y se deberán tener en cuanta en cada momento.

Servidor Web. Como se explicó anteriormente, toda la plataforma se basa en XML-RPC. Esta es la razón por la cual es necesaria una implementación de HTTP. Es una implementación muy simple y se puede desarrollar fácilmente o bien utilizar algún servidor web ya existente. Apache es el servidor web más común [4.4], tiene muchas opciones de seguridad, almacenamiento en caché, etc.., pero necesita una gran cantidad de recursos. Si la arquitectura hardware no es muy potente se deberán estudiar otras opciones como , *thttpd*, *cherokee* o *boa* [4.5].

Para validar la arquitectura propuesta en el presente trabajo, se han realizado diferentes implementaciones cada una basada en un hardware distinto, mostrando de esta manera tanto la robustez del estudio como la independencia del hardware defendida.

Las diferentes implementaciones realizadas y estudiadas se han basado en un elemento de control con unas características muy diferenciadas y han sido:

I. Basada en la placa base VIA EPIA TC 10.000
II. Basada en la plataforma GumStix
III. Basada en un sistema embebido X-Scale (NSLU2)
IV. Basado en FPGA

4.2. Implementación basada en la placa base VIA EPIA TC10.000

La primera arquitectura utilizada para implementar la plataforma GdRBot como un robot práctico se ha basado en una placa base de la empresa VIA Technologies Inc [4.6]. Todo el sistema se controla mediante una distribución de Linux tipo Debian [4.7]. Las rutinas del servidor se han implementado como CGIs en Apache. Las herramientas de desarrollo y depuración se han instalado de forma sencilla tanto dentro de la propia plataforma como fuera de ella. La placa base utilizada dispone de multitud de puertos de diferentes tipos para conectar todo tipo de sensores, disponiendo de las interfaces de comunicación necesarias.

Los elementos que forman el sistema son:

- Placa base modelo EPIA TC10.000.
- Motores paso a paso, ruedas y *driver* de control.
- Tarjeta de red inalámbrica por USB
- Servidor Sensorial GPBot: desarrollado en la tesis, consiste en una placa de desarrollo con un micro-controlador, conectado por puerto serie [4.8].
- Sensores de distancia, tanto por ultrasonidos como por infrarrojos
- Cámara USB
- Batería de plomo

4.2.1. Placa base: EPIA TC10.000

La placa base ultra compacta VIA EPIA TC Mini-ITX de 17 x 17 cm, figura 4.2, es una plataforma x86, altamente integrada y configurable, con un gran número de elementos que permiten su conectividad y opciones multimedia. Incluye conexión Ethernet, lo que permite el desarrollo de sistemas de red. El conjunto de dispositivos integrados en la propia tarjeta evita a los desarrolladores la necesidad de añadir componentes adicionales, reduciendo el costo y el tiempo de puesta en el mercado.

Figura 4.2. Fotografía de la placa VIA EPIA TC10.000.

La VIA EPIA TC está disponible con el procesador VIA Eden™ ESP, de ultra bajo consumo de energía y al no disponer de ventilador los diseños son silenciosos, o el procesador VIA C3™ para aplicaciones multimedia de uso intensivo. El rendimiento multimedia se mejora gracias al chipset VIA CLE266, con integrados VIA UniChrome™ para gráficos 2D/3D y acelerador MPEG-2 con compensación de movimiento, además de soporte para memorias DDR266.

4.2.2. Motores Paso a Paso, ruedas y driver de control

Como sistema de tracción para esta implementación, se ha diseñado y fabricado un conjunto robusto formado por dos motores paso a paso, dos ruedas de goma, un par de engranajes con una correa dentada y una estructura metálica que lo une todo (figura 4.3).

El motor paso a paso utilizado es el modelo T180 de Astrosyn y las características principales de estos motores son:

- Conexión de las bobinas: Unipolar / Bipolar
- Rango de corriente:0,6A
- Número de fases:2

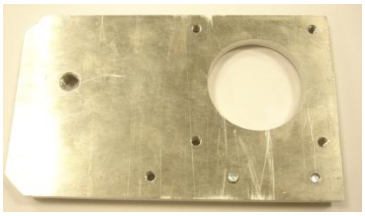
- Resistencia:20ohm
- Inductancia:35mH
- Inercia:110g-cm²
- Tamaño del marco: NEMA 23
- Par de mantenimiento: 0.539N-m
- Grado de giro:1.8°
- Tensión de funcionamiento: 12V DC

La ventaja del uso de este tipo de motor es la precisión que se consigue en los desplazamientos, aunque en contra está el mayor consumo y mayor peso respecto del equivalente en motores de continua.

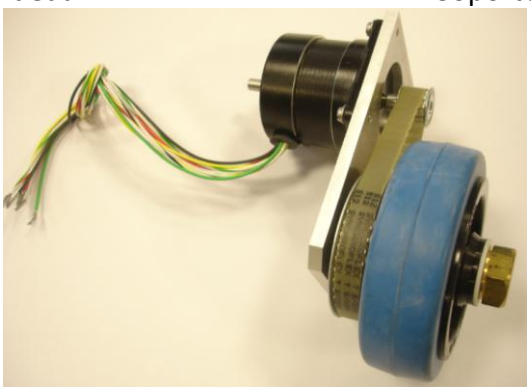
Motor Paso a Paso, (MY180 de Astrosyn), añadida polea de 10 dientes

Rueda de goma (modelo de TENTE), añadida polea de 50 dientes

Eje de la rueda



Soporte metálico



Conjunto rueda-motor completo montado

Figura 4.3. Sistema motriz de la plataforma

La rueda es el modelo UFP100x34-Ø8 blue, de TENTE. El núcleo de la rueda es de poliamida, con la banda de rodadura de goma elástica y cojinete de bolas de precisión.

La unión entre el motor y la rueda se realiza a través de un conjunto de dos poleas y una correa de distribución dentadas. El tamaño diferente de las poleas, con una relación 1:5, permite aumentar aún más el par del motor, consiguiendo una mayor fuerza en su conjunto, por supuesto en detrimento de la velocidad que, en la gran mayoría de la aplicaciones, no es un requisito primario.

Para unirlo todo, y dentro de los desarrollos de la tesis, se han diseñado y fabricado también las piezas metálicas de unión correspondientes, así como el eje de la rueda.

Gran parte del éxito en el resultado obtenido por un determinado motor, reside en el driver que se utilice para su control. Casi tan importante como la elección del tipo de motor a usar es la elección del controlador del mismo. En este apartado, como en el resto de los temas tratados en esta tesis, se ha intentado mantener la idea de sistemas distribuidos y autónomos. Se pretende evitar que exista un controlador central que cargue con todas las tareas. Por este motivo el control de cada motor se lleva a cabo por un microcontrolador especializado. Esto no sólo libera de trabajo al procesador central sino que incorpora una serie de utilidades que permiten poder adaptar la velocidad, aceleración, consumo, etc..., a cada situación de una manera muy sencilla y completa. La conexión entre la unidad central y los controladores de los motores se realiza a través de un puerto USB.

El *driver* elegido es el *Pilot Motion Processor* MC3410, de PMD [4.9]. Es un procesador que ofrece funciones de generación de trayectorias y control relacionado con los motores paso a paso. Las características principales de este dispositivo se muestran en la tabla 4.1.

Para el control de la posición dispone de unas entradas que le informan de la situación del eje. Si se utiliza una señal incremental se filtran digitalmente dos señales de entrada en cuadratura y se pasa a un contador ascendente/descendente rápido. Usando la interfaz de entrada de 16 bits el chip lee la posición codificada en binario.

También dispone de un generador de trayectorias que calcula la nueva posición deseada en cada intervalo basándose en el modo seleccionado y los parámetros

introducidos. El intervalo coincide con los instantes de tiempo en los que el sistema actualiza la trayectoria, la compensación de los motores y otras funciones del chip.

Tabla 4.1.Características del driver MC3410

Configuración	Un único eje
Modo de operación	En lazo abierto
Modos de Comunicación	Paralelo 8/16 bits (8 bits en el bus externo y 16 en bits de tamaño del comando interno
	Paralelo 16/16 bits (16 bits en el bus externo y 16 en bits de tamaño del comando interno
	Serie asíncrono punto a punto
	Serie asíncrono multipunto
Velocidad del puerto serie	Desde 1.200 a 416.667 baudios
Tipos de trayectorias	Trapezoidal punto a punto. Los parámetros son velocidad, aceleración, jerk y posición.
	Curva en "S" punto a punto. Los parámetros son velocidad, aceleración, deceleración y posición.
	Seguimiento de velocidad. Los parámetros son velocidad, aceleración y deceleración.
Modos de control del motor	PWM (80KHz, 8 bits) y DAC (16 bits)

La salida del generador de trayectorias se combina con la posición actual del *encoder* para calcular el error de posición, con 32 bits, el cual sirve de realimentación a un controlador PID. El resultado se amplifica y sale del chip como una señal PWM o como una señal analógica.

La comunicación del chip con el procesador central se realiza a través de una comunicación serie. Se utilizan comandos cortos enviados y recibidos como secuencias de bits o palabras. Estos paquetes contienen el código de instrucción que indica el tipo de acción a realizar. Pueden contener datos, tanto en el envío como en la respuesta.

Una vez seleccionado este *driver* de motores apareció un problema, y fue que no existía ningún código de uso escrito para Linux. Es decir, todas las librerías que ofrecía el fabricante sólo se podían utilizar desde el sistema operativo Windows. Por tanto, y dado que este dispositivo se había elegido entre una amplia oferta por ser el único que reunía las prestaciones necesarias, como parte de este trabajo de tesis se escribió una

librería completa para el acceso desde Linux. En el Anexo III se incluye la lista de comandos implementados y un fichero de cabecera de c (librobot.h) a modo de ejemplo del código escrito.

Debido a que el generador de trayectorias es uno de los motivos por los que se seleccionó este *driver*, y la librería escrita para Linux se centra en sacar el máximo partido es este punto, a continuación se van a describir brevemente los diferentes modos de los que dispone:

- Trapezoidal punto a punto
- Curva en "S" punto a punto
- Seguimiento de velocidad

Trapezoidal punto a punto. El controlador central especifica una aceleración inicial, velocidad, deceleración y posición final. El eje del motor se acelera linealmente, según el parámetro introducido, hasta que consigue la velocidad seleccionada. Continúa en movimiento a esa velocidad y decelera linealmente hasta detenerse en la posición especificada

Si la deceleración llega antes de haber conseguido la velocidad programada, no se llegará al segmento de velocidad constante y en lugar de un trapezoide habrá un triángulo.

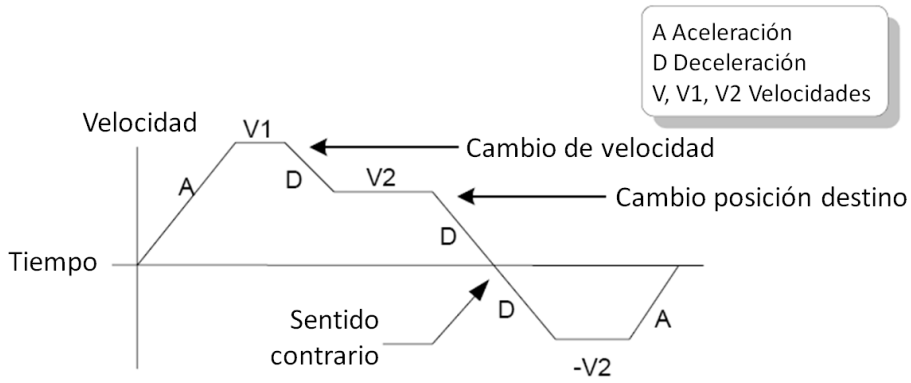


Figura 4.4. Trayectoria punto a punto trapezoidal compleja.

Se pueden modificar parámetros durante el movimiento del motor. El generador de trayectorias siempre intentará cumplir con los parámetros introducidos. Por ejemplo, si se introduce una nueva posición de destino que ya ha sido rebasada, el generador

decelera hasta parar y cambia el sentido de giro, haciendo de la deceleración la aceleración del sentido contrario. Esta situación se refleja en la figura 4.4.

Curva en "S" punto a punto. La trayectoria de curva en "S" punto a punto agrega un límite a la tasa de cambio de la aceleración de la curva trapezoidal básica. Se añade un nuevo parámetro, *jerk* (en inglés "tirón"), que especifica el cambio máximo en la aceleración en un solo ciclo. Esto impide que la aceleración cambie bruscamente y con ello el par de fuerza. De esta manera no se pierden pasos en el motor, conservando los valores de control de pasos que sirven para medir la distancia recorrida.

En este modo, la aceleración se incrementa gradualmente desde 0 hasta el valor de la aceleración programada y disminuye al mismo ritmo hasta llegar a 0 de nuevo cuando se alcanza la velocidad programada. La misma secuencia a la inversa detiene el eje al llegar a la posición de destino seleccionada.

La figura 4.5 muestra una típica curva del modo en "S". En el segmento I, el perfil de la curva S lleva al eje, al *jerk* especificado (J), hasta que se consigue la máxima aceleración (A). El eje continúa su aceleración lineal (*jerk*=0) hasta el segmento III. En este segmento se vuelva a aplicar el valor negativo del *jerk* para reducir la aceleración hasta cero. Así se llega al segmento IV, en el que se alcanza la máxima velocidad.

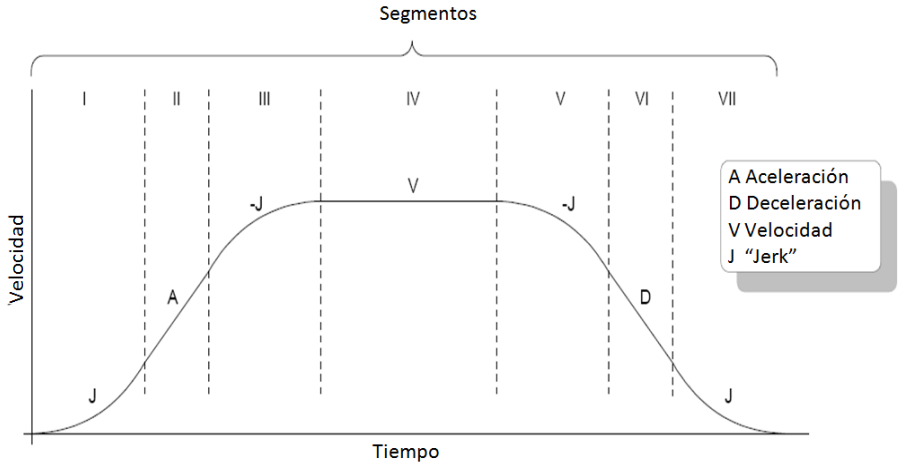


Figura 4.5. Trayectoria curva en S.

Para reducir la velocidad, la curva sufre una desaceleración de manera similar a la fase de aceleración, utilizando en primer lugar el valor de *jerk* para alcanzar la máxima deceleración (D) y luego para que el eje llegue a su fin en el destino.

Puede haber situaciones en los que una trayectoria en S no contenga todos los segmentos mostrados en la figura anterior.

Seguimiento de velocidad. A diferencia de los modos de perfil trapezoidal y de curva en S, en donde la posición de destino determina la dirección del viaje inicial, en el modo de perfil de Seguimiento de Velocidad el signo del parámetro de velocidad determina la dirección inicial de movimiento. Por lo tanto el valor de velocidad que se envía al *chipset* puede tener valores positivos, para el movimiento de sentido positivo, o valores negativos para el movimiento contrario.

En este perfil, no se especifica la posición de destino. El movimiento es controlado en su totalidad por el cambio de la aceleración, la velocidad y los parámetros de desaceleración, mientras que el perfil está siendo ejecutado.

La trayectoria se ejecuta por una continua aceleración en el eje, a la tasa establecida, hasta que se alcanza la velocidad seleccionada. El eje comienza la desaceleración cuando se especifica una nueva velocidad, siempre que sea menor en magnitud que la velocidad actual, o tiene una señal de que es opuesta a la actual dirección del viaje.

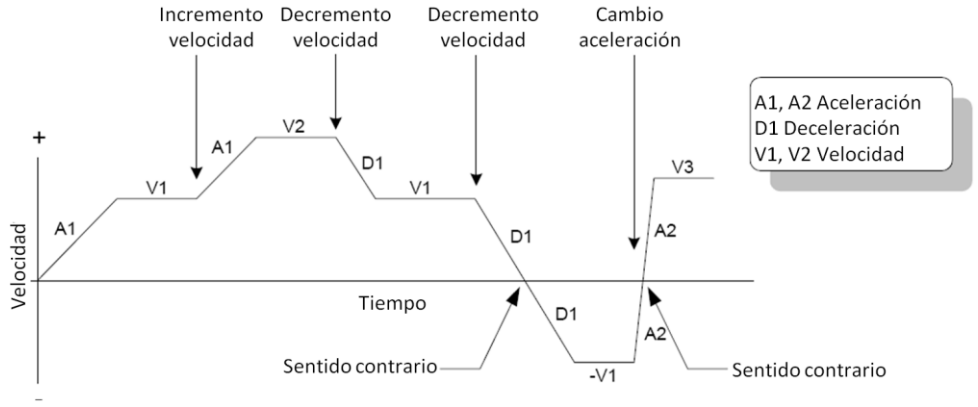


Figura 4.6. Trayectoria de Seguimiento de Velocidad.

Un perfil simple de Seguimiento de Velocidad se parece a un perfil Trapezoidal Punto a Punto, como el mostrado en la figura 4.4. La figura 4.6 muestra un perfil más complejo, en el que tanto la velocidad como la dirección cambian dos veces.

4.2.3. Tarjeta de red inalámbrica

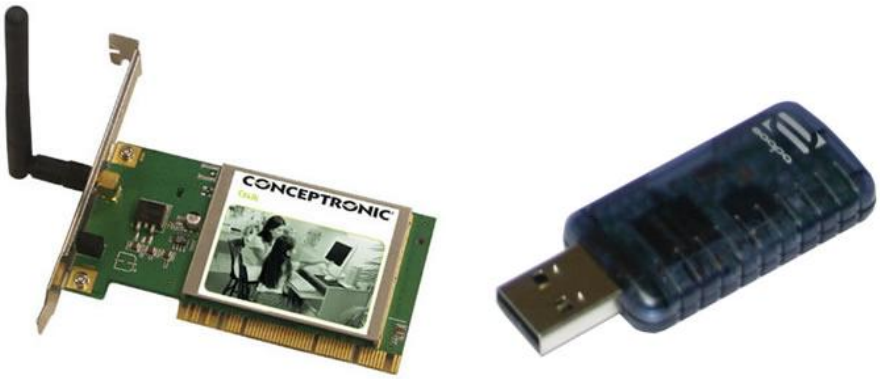
En esta implementación de la plataforma, para realizar la comunicación se ha utilizado una red Ethernet, usando el estándar IEEE802.11.

Para ello, en la primera versión implementada de la plataforma se utilizó una tarjeta de red PCI, en concreto el modelo C54Ri de Conceptronics, figura 4.7a, cuyas principales características son:

- Soporte para Microsoft Windows 98SE, Me, 2000 and XP y para Linux
- Permite encriptación WEP 64; WPA; WEP 128; WPA2
- Opera en la banda libre de frecuencia ISM de 2.400GHz ~ 2.4835GHz.
- Métodos de modulación:
    - IEEE 802.11b: DSSS (*Direct Sequence Spread Spectrum*).
    - IEEE 802.11g: OFDM (*Orthogonal Frequency Division Multiplexing*).
- Velocidad 54Mb/s

En la segunda versión se ha utilizado un adaptador de red USB a WIFI 54Mbps, figura 4.7b. En concreto, se ha utilizado el modelo *WLAN 11g USB adapter*, de la marca Zaapa, cuyas principales características son:

- Soporte para Microsoft Windows 98SE, Me, 2000 and XP y para Linux
- Alcance en interiores entre 35m y 100m y hasta 300m al aire libre.
- Velocidad seleccionable: 54/48/36/24/18/12/11/9/6/5.5/2/1 Mbps.
- Soporta interface USB 2.0.
- Permite encriptación WEP de 64/128 bits y WPA
- Opera en la banda libre de frecuencia ISM de 2.400GHz ~ 2.4835GHz.
- Métodos de modulación:
    - IEEE 802.11b: DSSS (*Direct Sequence Spread Spectrum*).
    - IEEE 802.11g: OFDM (*Orthogonal Frequency Division Multiplexing*).



(a) **(b)**

Figura 4.7. Tarjetas de red Wifi utilizadas

4.2.4. Servidor sensorial GP_Bot

Como ya se comentó con anterioridad, el objetivo del Servidor Sensorial es quitar trabajo al procesador central, resolviendo los accesos de bajo nivel a los diferentes sensores que se puedan conectar a la plataforma. Para ello se utiliza un sencillo micro-controlador dotado del canal de comunicaciones apropiado para su conexión con dicho controlador central. Se ha definido una interfaz software para la comunicación entre la Unidad Central de Control y el Servidor Sensorial para que el Sistema se pueda ampliar con cualquier Servidor Sensorial, siempre que utilice la interfaz correcta. Para los Servidores Sensoriales desarrollados dentro de la tesis se ha diseñado un sencillo gestor de servicios, de forma común a todos. De esta manera se puede conectar cualquier tipo de sensor o actuador gestionados por un micro-controlador, sin más que resida en él un gestor de servicios similar que atienda las mismas peticiones con igual interfaz. Este Servidor Sensorial ha sido diseñado y fabricado dentro del presenta trabajo de tesis. Para obtener todos los detalles, en el Anexo I se puede consultar su Manual de Usuario, con el detalle de todos los conectores y esquemas eléctricos.

Como especificación inicial se estableció que el sistema debería ser flexible, de propósito general y bajo costo. Debería contar con los recursos suficientes para el control de elementos básicos utilizados en un robot sencillo, tales como: sensores de infrarrojos, ultrasonidos, pulsadores, motores y servomecanismos y sistemas de comunicación por radio. El diseño debería tener en cuenta recursos para el control de dispositivos más complejos, que requieran de capacidad de cálculo y de memoria suficiente para gestionar algoritmos avanzados y otras comunicaciones.

El módulo de radio de AM, figura 4.8, es un circuito híbrido de la empresa Aurel [4.10] para transmisión y recepción de datos digitales con antena única. Permite comunicación “half duplex”. Utiliza el rango de frecuencia para radiomandos y radiocontrol de 433,92 MHz, banda de uso libre. La antena está formada por un hilo de cable conductor de 17 cm, con un alcance de unos 200 metros sin obstáculos.

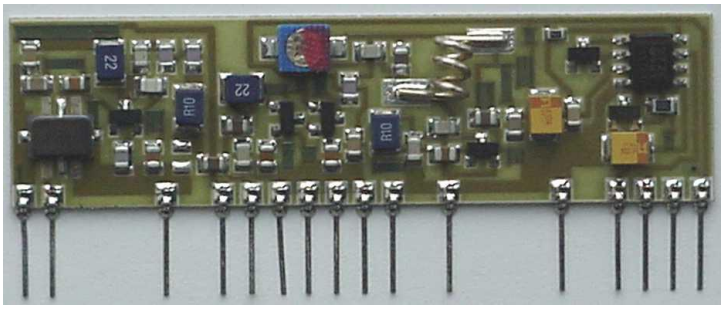


Figura 4.8. Módulo de radio de Aurel

El Servidor Sensorial GP_Bot, mostrado en la figura 4.9, está formado por un conjunto de dos tarjetas de igual tamaño, GP_Bot y GP_Bot_Ifaz, que se interconectan entre sí, más un programador externo, GP_Mon, que además permite la depuración on-line de los programas instalados.

4.2.4.1. La Tarjeta GP_Bot

La tarjeta GP_Bot es una placa de desarrollo de propósito general basada en el microcontrolador de 8 bits *MC68HC908GP32* de Motorola [4.11]. Su arquitectura basada en la familia MC68HC08 [4.12] está optimizada para compiladores C y aplicaciones de control. Este micro es completamente estático e incluye opciones de diseño de bajo consumo. Puede trabajar en modo Monitor, a través del cual se puede acceder desde un PC a todos los recursos internos, registros y contenidos de memoria, tanto ROM (*Flash*) como RAM, permitiendo incluso la ejecución paso a paso del programa almacenado.

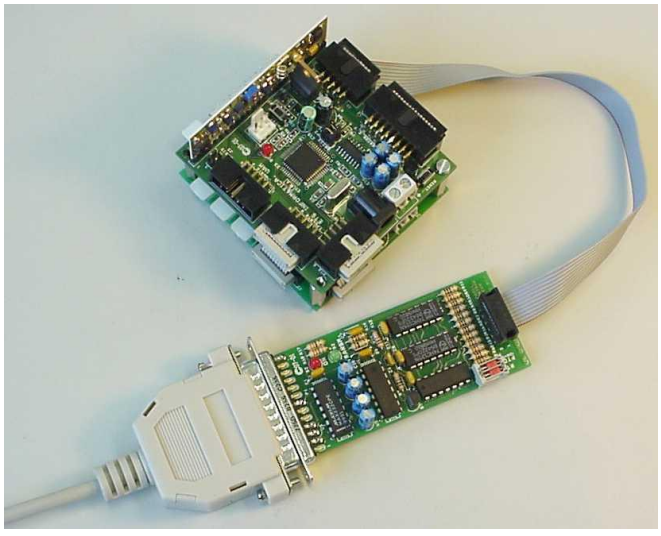


Figura 4.9. Imagen del Servidor Sensorial GP_BOT completo

Se eligió un encapsulado QFP *(quad flat pack*) de 44 *pines*, que ha permitido que la tarjeta tenga unas reducidas dimensiones (70 x 65 mm) y el máximo número de E/S de usuario. A partir de este micro-controlador, se construyó una tarjeta con las siguientes características:

- Micro-controlador MC68HC908GP32, con un cristal de 9.8304 MHz.
- Señales de *reset* y entrada de interrupción. El *reset* permite sincronizar el arranque de todos los componentes conectados y las entradas de interrupción pueden gestionar eventos en cuanto se producen.
- Pulsadores de *Reset* y *Power-On*. Son necesarios para inicializar la aplicación que se está ejecutando y/o permitir la reprogramación del microcontrolador.
- *Drivers* serie tipo RS-232 (MAX232 [4.13]) y RS-485 (MAX3082 [4.13]). *Drivers* hardware que permiten una comunicación serie RS-232 y RS-485 respectivamente.
- Módulo para transmisión-recepción AM por radio, en la banda de los 433 MHz. [4.10]
- Fuente de alimentación regulada de 5v. Genera una tensión fija y estabilizada que protege al micro-controlador y al resto de componentes del sistema de desarrollo.
- Entrada de cable monitor para la conexión desde PC, según las especificaciones del software de desarrollo utilizado Codewarrior Development Tools, de la empresa Freescale [4.14]. Puerto para la conexión del programador externo.
- Acceso externo a todos los *pines* del micro-controlador: Puerto A (8 bits de E/S o conexión de un teclado), Puerto B (8 bits E/S o canales al conversor Analógico-Digital), Puerto C (7 bits E/S), Puerto D (8 bits E/S o temporizadores y puerto serie síncrono) y Puerto E (2 bits de E/S o puerto serie asíncrono). Conectores normalizados para la conexión con el exterior.

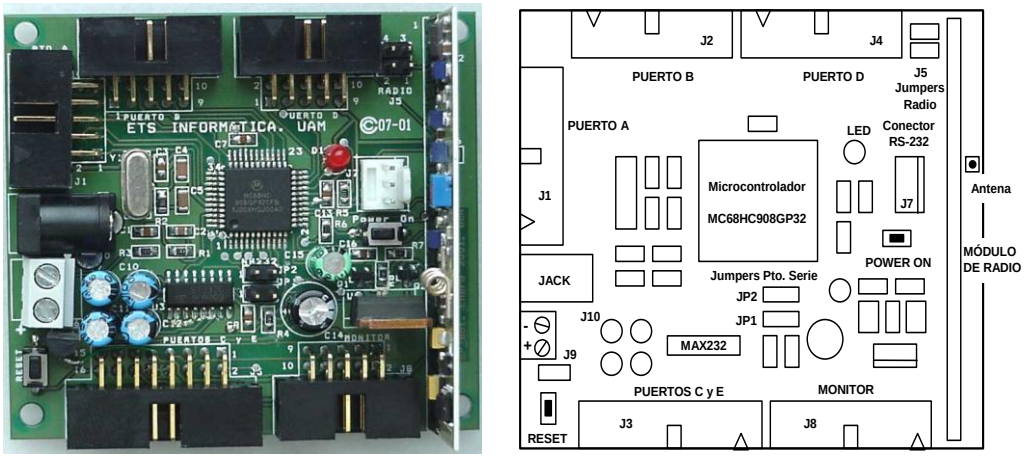


Figura 4.10. Tarjeta GP_Bot (a). Esquema de componentes (b)

4.2.4.2. Tarjeta GP_Bot_Ifaz

Esta tarjeta ha sido diseñada para actuar como interface entre la tarjeta GP_Bot y cierto tipo de actuadores, tales como motores, sensores de infrarrojos y pulsadores. También dispone de una serie de *pines* de E/S para otras aplicaciones del diseñador. El tamaño de esta tarjeta es idéntico al de la GP_Bot y la colocación de los conectores coincide entre ambas tarjetas, permitiendo crear un conjunto pequeño y compacto al ser conectadas a través de un conjunto de cables en formato cinta plana. Sus características principales son:

- Manejo de hasta 4 motores de corriente continua o dos motores paso a paso.
- Conexión directa para 4 sensores de infrarrojos, incluyendo su polarización.
- 8 entradas analógicas que también pueden configurarse como E/S digitales.
- 4 señales de E/S digital.
- Permite alimentar los motores con tensiones diferentes a la de la lógica.
- Fuente de alimentación regulada, con filtro LC.

El *driver* de motores está basado en dos integrados tipo L293D [4.15], mientras que el *driver* de infrarrojos está diseñado para conectar hasta cuatro sensores CNY70 [4.16] o equivalentes. Incluye las resistencias de excitación de los emisores de infrarrojos y las de *pull-up* (internas al micro-controlador). Cada sensor de infrarrojo se puede leer desde un puerto digital o desde uno analógico. Esto último permite al sistema distinguir entre varios colores.

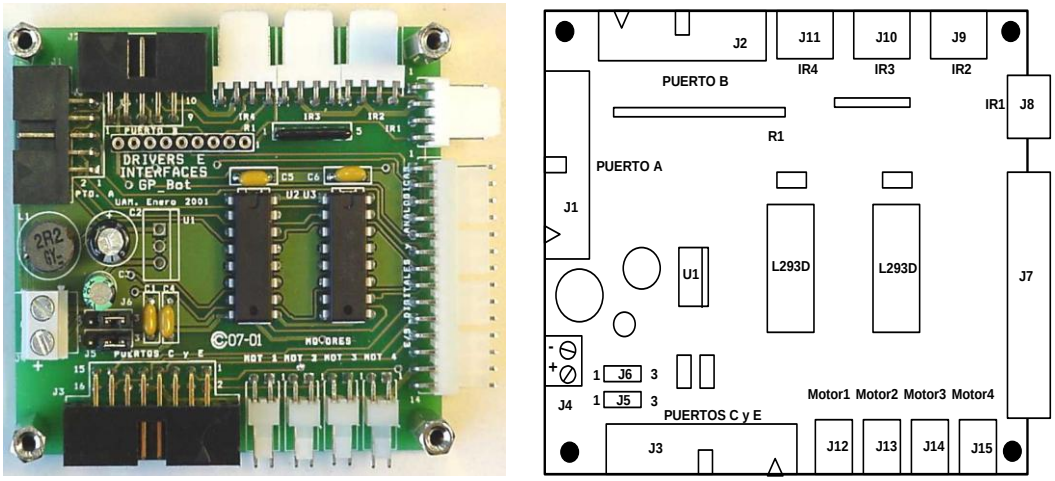


Figura 4.11. Tarjeta GP_Bot_Ifaz (a). Esquema de componentes (b)

4.2.4.3. Tarjeta GP_Mon

El sistema se programa desde un PC, mediante el puerto serie, accediendo al potente Modo Monitor del micro-controlador, permitiendo programar y depurar cualquier aplicación.

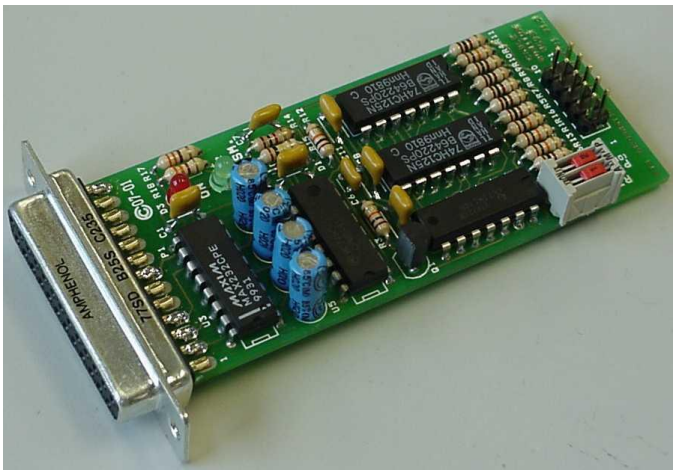


Figura 4.12. Imagen de la tarjeta GP_Mon

Para utilizar las funciones del Monitor se ha desarrollado esta tarjeta, figura 4.12, que contiene los circuitos necesarios para la adaptación de niveles del puerto serie, y los circuitos necesarios para forzar la entrada del micro-controlador en dicho modo.

4.2.5. Sensores de Distancia: Por ultrasonidos y por infrarrojos

Para evitar colisiones, la plataforma dispone de un conjunto de sensores que miden la distancia a la que se encuentra un obstáculo. De esta forma se evitan los choques, tanto con elementos sueltos que puede haber en el camino como con paredes.

Se han utilizado dos tipos se sensores, unos basados en ultrasonidos y otros en emisiones infrarrojas.

4.2.5.1. Sensores de ultrasonidos SRF04 / SRF05

Para calcular la distancia por ultrasonidos se han empleado, por su extendido uso y popularidad, los sensores comerciales SRF04 y SRF05 indistintamente. Ambos sensores contienen toda la electrónica encargada de hacer la medición y tienen un funcionamiento muy sencillo (Figura 4.13)

Figura 4.13. Sensor de ultrasonidos SRF04/05

Para realizar la lectura, el micro-controlador del Servidor Sensorial debe emitir un pulso de anchura mínima de 10µs por el pin "Disparo" del sensor (Entrada Eco en el SRF05 y Entrada Disparo en el SRF04).

A continuación, el módulo emite una ráfaga de ocho impulsos ultrasónicos a una frecuencia de 40 kHz., dicha ráfaga choca contra el objeto y rebota siendo capturada de nuevo por el sensor.

Finalmente, por el pin "Salida Eco" del sensor aparece un pulso cuya duración es proporcional a la distancia entre el sensor y el objeto.

Puede realizarse de nuevo todo el proceso para tomar otra medida de la distancia teniendo en cuenta que hay que dejar pasar un tiempo mínimo dependiendo de si se usa uno u otro modelo para que el circuito se estabilice y la señal de eco desaparezca completamente.

A continuación, en la figura 4.14, se muestra un diagrama de tiempos del funcionamiento extraído de la página del fabricante:

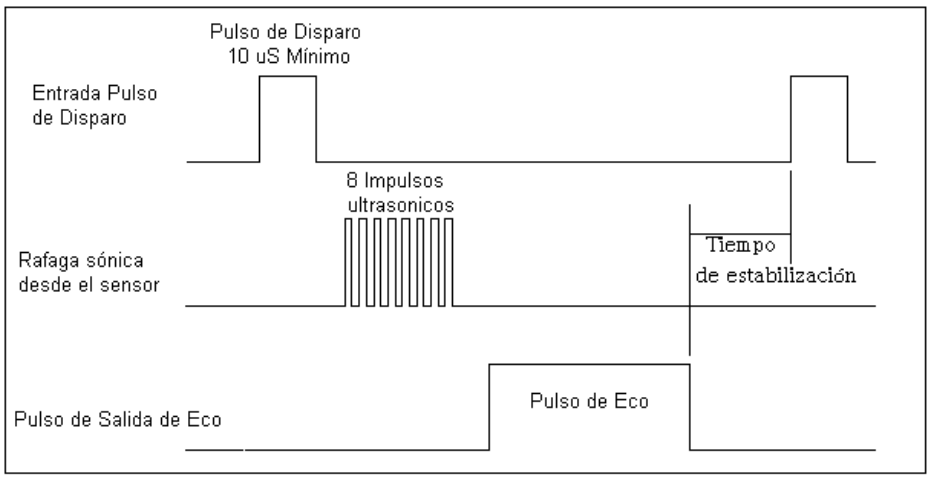
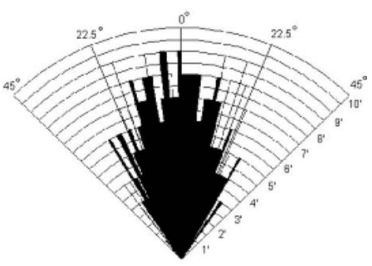
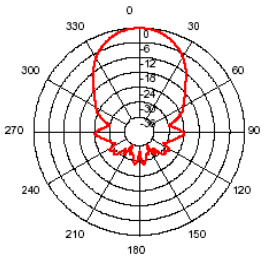


Figura 4.14. Diagrama de tiempos SRF04 / SRF05

Dependiendo del modelo que se use, existen algunas particularidades que se muestran en la tabla 4.2.

Tabla 4.2. Comparativa entre sensores de ultrasonidos SRF04 y SRF05

	SRF04	**SRF05**
Rango de trabajo	De 3 cm. a 3 m.	De 3 cm. a 4 m.
Rango de anchura del eco	De 100µs. a 18ms.	De 100µs. a 25 ms.
Espera entre fin de eco y disparo	10ms.	50ms.
Ancho del eco si no detecta objeto	36ms.	30ms.
Anchura del haz		

Para conseguir compatibilidad entre los sensores SRF04 y SRF05 se mantiene una espera de 50ms. entre la finalización del eco actual y el comienzo del disparo siguiente y se considera que no hay obstáculo si la anchura del pulso de eco es superior a 30ms.

4.2.5.2. Sensores de infrarrojos SHARP GP2D12

Para calcular la distancia por infrarrojos se ha empleado el sensor comercial GP2D12 de Sharp (Figura 4.15).

Figura 4.15. Vista frontal GP2D12

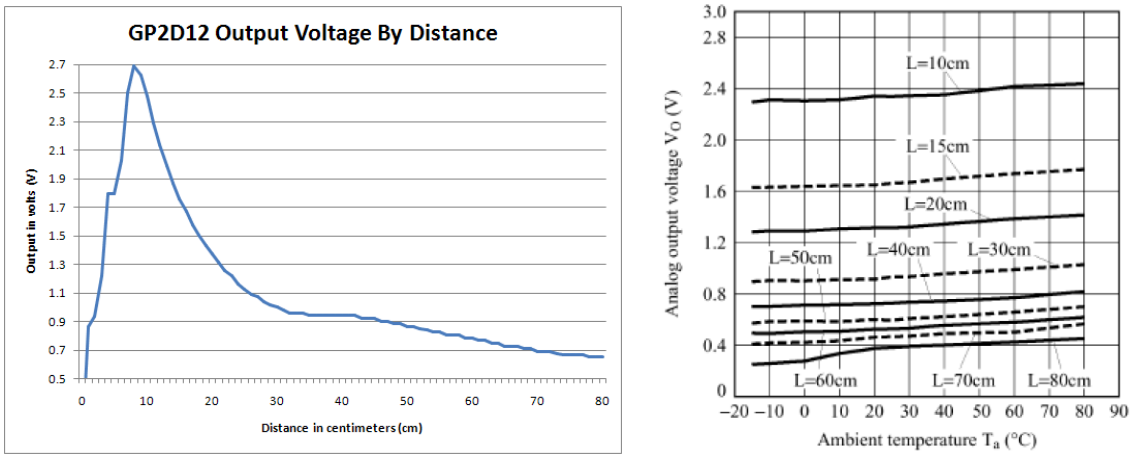
En la parte frontal del sensor se pueden distinguir dos lentes. En la parte interior de la lente de la derecha hay un LED infrarrojo. En la parte interior de la lente de la izquierda hay lo que se conoce como PSD [4.17] o sensor detector de posición.

El sensor dispone de tres pines; uno es el de alimentación (4.5V-5.5V), otro es el de referencia de tensión (0V) y el tercero es el de salida (V_0).

Según el fabricante es necesario situar un condensador de desacoplo entre el pin de alimentación y el pin de masa de valor 10 µF como mínimo para estabilizar la tensión de referencia del sensor.

Las medidas de distancia que pueden tomarse con el sensor GP2D12 varían de 10cm. a 80cm y se determinan en base a la tensión analógica que entrega el pin de salida.

A continuación se muestran dos gráficas, una que ilustra la tensión analógica del pin de salida en función de la distancia a que se encuentra el objeto (Figura 4.16a) extraída de [4.18] y otra que ilustra la sensibilidad del sensor con respecto a la temperatura para diferentes distancias (Figura 4.16b) extraída de [4.19]).



(a) **(b)**

Figura 4.16. (a) Sensibilidad vs distancia. (b) Sensibilidad vs Temperatura

Como se puede observar en la figura 4.16a, la tensión que entrega a su salida el sensor de infrarrojos hace que este sea un elemento altamente no lineal. Es por ello que para convertir la tensión analógica V_0 a distancia en cm. es necesario el uso de una tabla de conversión que debería ser actualizada mediante una calibración cada vez que cambia el entorno en el que se mueve la plataforma. De la figura 4.16b se deduce la influencia de la temperatura sobre la medida entregada por el sensor. Obsérvese que no existe una correlación directa entre el error debido a la temperatura y la distancia que se está midiendo.

4.2.6. Conjunto montado

Para finalizar este apartado, se presentan unas imágenes que muestran varias de las implementaciones realizadas uniendo todos los elementos descritos para la plataforma basada en la placa base de VIA EPIA, así como algunos detalles más.

En la figura 4.17 se muestra una de las implementaciones de la plataforma basada en la placa VIA EPIA TC 10.000. Sobre una estructura de aluminio se han montado sendos motores paso a paso con sus correspondientes ruedas. Sobre la plataforma se puede ver una cámara de video montada una estructura móvil gobernada por dos servomotores. Éstos permiten mover la cámara en los planos vertical y horizontal según las necesidades de la aplicación. Es decir, pueden “mirar” de arriba a abajo y de izquierda a derecha. En la parte derecha de la imagen se puede ver una pantalla LCD,

conectada a un Servidor Sensorial que la controla así como a una serie de pulsadores e interruptores. En la pantalla se pueden mostrar los diferentes mensajes del estado del robot, valores de medida se sus sensores, etc.

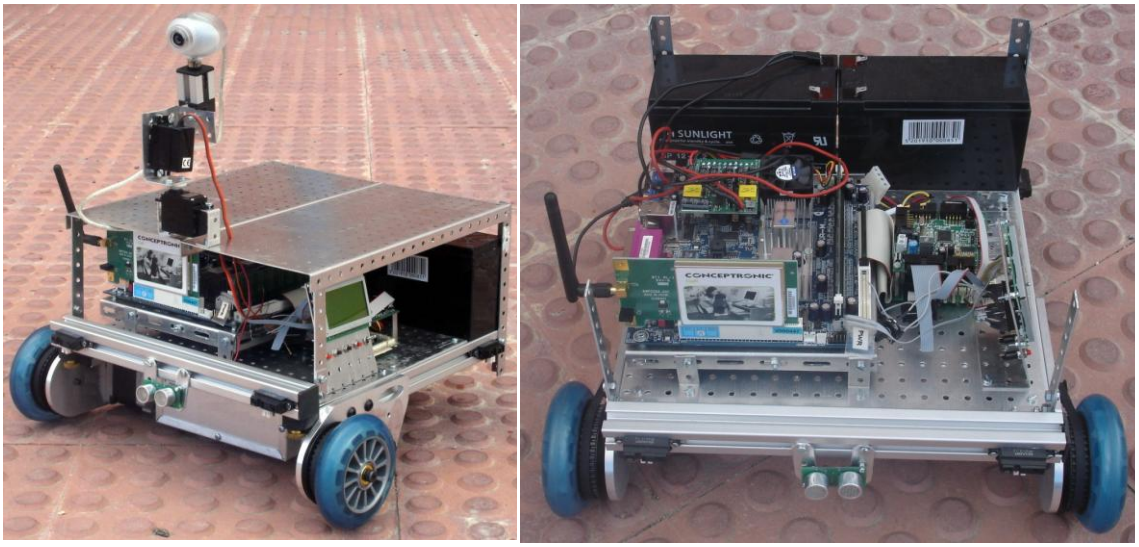


Figura 4.17. GdR_Bot. Primera implementación.

En la parte frontal se distinguen tres sensores de distancia, los de los lados son de infrarrojos y el del centro de ultrasonidos. Esto permite una detección de obstáculos desde pocos centímetros hasta algunos metros.

En el lateral izquierdo, atrás, hay otro sensor de distancia por infrarrojos. Éste se utiliza para desplazarse de forma paralela a una pared lateral. En la derecha dispone de otro igual.

En la foto de la derecha de la figura 4.17, se muestra el detalle del interior (sin tapa) de esta implementación. Además de lo mencionado anteriormente, se puede ver la placa VIA EPIA, en la izquierda de la imagen, con un tarjeta de red inalámbrica conectada, así como el Servidor Sensorial GP_Bot que realiza el control de los sensores y del LCD. Al fondo se encuentran dos baterías de plomo, que se conectan en paralelo para dotar de más autonomía al sistema.

En la foto de la izquierda de la figura 4.18 se observa el detalle del montaje de la cámara de vídeo. La cámara está sujeta a una estructura gobernada por dos servomotores. Éstos permiten girarla tanto en un plano vertical (servomotor superior) como horizontal (servomotor horizontal). Estos servomotores están controlados a

través del Servidor Sensorial. En la foto de la derecha de la misma figura 4.18, se observa el detalle de la conexión del Servidor Sensorial a una pantalla LCD que permite visualizar de forma continua la actividad del mismo.

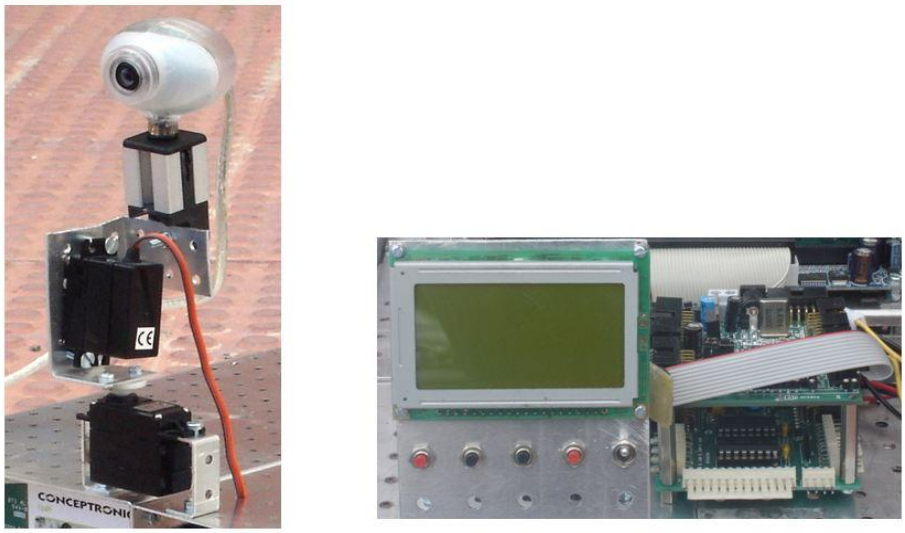


Figura 4.18. Detalle de la cámara y del Servidor Sensorial

Por último en la figura 4.19 se muestra una segunda implementación basada también en la misma placa VIA EPIA, muy similar en características a la primera en la que básicamente sólo cambia la estructura de soporte y que se la ha añadido un segundo Servidor Sensorial para poder realizar la gestión de más sensores y actuadores.

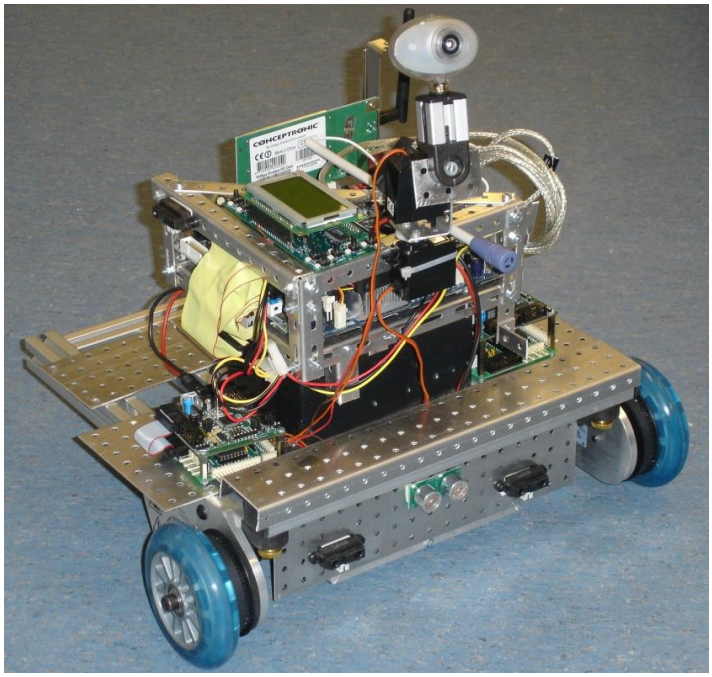


Figura 4.19. GdRBot. Segunda implementación.

Como nexo de unión entre todos los componentes, se han utilizado diferentes perfiles de aluminio con los conectores apropiados.

4.3. Implementación basada en el sistema de desarrollo GumStix

Pese a que la plataforma anterior cumplía satisfactoriamente los objetivos propuestos, el tamaño era excesivamente grande para ciertas aplicaciones, limitado por las dimensiones de la propia placa VIA EPIA y de las baterías que necesitaba.

Por ello se acometió una nueva implementación física, que por una parte servía para demostrar la generalidad de la plataforma y por otra se buscaba reducir el tamaño de la anterior. Por lo tanto la principal característica de esta implementación es su reducida dimensión.

La nueva implementación se basa en el sistema de desarrollo Gumstix, sistema de dimensiones muy pequeñas que cumple con las características mínimas necesarias para poder implementar sobre él la plataforma GdR_Bot y lo suficientemente potente para la ejecución de múltiples aplicaciones.

Figura 4.20. Conjunto montado del Sistema Gumstix.

Los elementos que forman esta implementación son (figura 4.20):

- Placa base modelo Gumstix. Sistema de desarrollo basado en el microprocesador
- Motores de corriente continua con encoder, tracción mediante un sistema de orugas tipo tanque y driver de control.

- Tarjeta de red inalámbrica por Bluetooth.
- Servidor Sensorial Robostix. Tarjeta de desarrollo basada en un micro-controlador de ATMEL ATMEGA128. Dispone de convertidores A/D, salidas PWM, líneas serie, entradas y salidas digitales, etc.
- Sensores de distancia por ultrasonidos.
- Cámara USB.
- Batería de plomo.

4.3.1. Descripción del Sistema Gumstix

La tarjeta base del sistema de desarrollo Gumstix [4.20] figura 4.21, está basado en el procesador Intel XScale PXA255 [4.21]. Su tamaño es de 80x20x6,3 mm. Su consumo máximo es de 250 mA y cuenta con conectividad inalámbrica vía *Bluetooth*. Dispone de diferentes ampliaciones, que incluyen Ethernet y Wifi. También se le puede conectar una tarjeta de expansión, denominada Robostix, que integra un micro-controlador de Atmel, el ATMEGA128, que puede ser utilizado para controlar diferentes tipos de sensores.

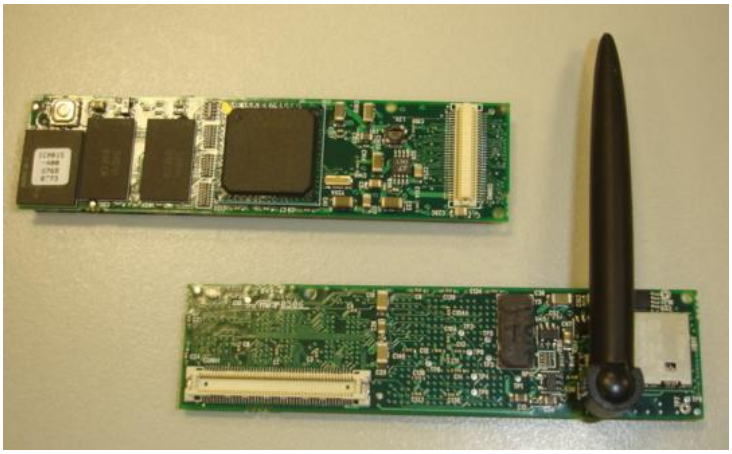


Figura 4.21. Tarjeta Gumstix

El sistema de desarrollo Gumstix ya cuenta con el sistema operativo Linux instalado [4.22]. De esta manera, la única tarea realizada ha sido la instalación del compilador cruzado en la máquina que actúa como *host* y la recopilación de los componentes del servidor. La conectividad se realiza a través de una interfaz de red tipo *bluetooth*.

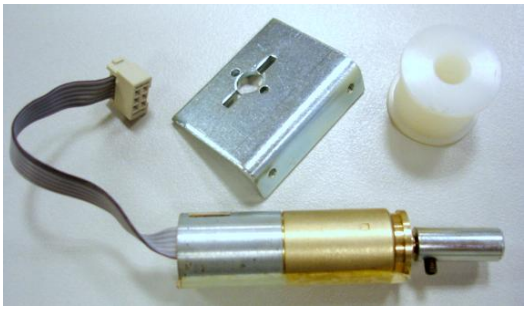
Las restricciones de este sistema es que no tiene un procesador muy potente. Si se tienen que ejecutar algoritmos complejos, éstos se deben ejecutar en la máquina cliente. Otra restricción es que no soporta muchas consultas por segundo.

Este sistema de desarrollo ha sido utilizado por numerosos grupos de investigación, lo que da una idea de su potencial [4.23], [4.24] y [4.25].

4.3.2. Motores de corriente continua, orugas y driver

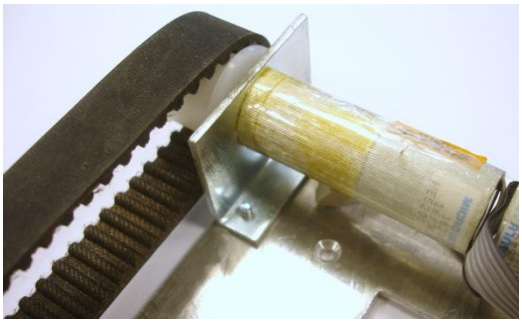
Para esta implementación, que lo que primaba era el tamaño, se han utilizado dos motores de corrientes continua modelo Faulthaber. Además de reducido tamaño y muy bajo consumo, su principal ventaja es que dispone de un *encoder* óptico, que entrega 128 pulsos por vuelta, en cuadratura.

Por probar otro tipo de tracción, se han diseñado y construido una serie de elementos para dotar de un mecanismo de oruga como sistema de tracción, similar al utilizado en tanques. En la figura 4.22 se muestra un desglose de las piezas que han sido fabricadas. En al Anexo V se incluyen todas las cotas de estas piezas.

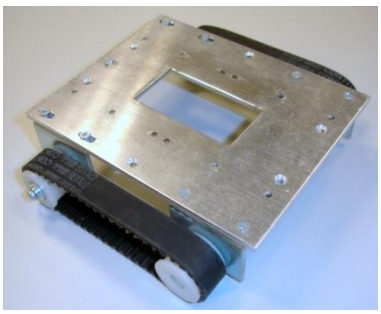


Motor, polea y soporte

Correas



Detalle motor-polea-correa



Sistema completo montado

Figura 4.22. Sistema de tracción completo

En este caso, dado el carácter más simple del desarrollo, el *driver* utilizado para el control de los motores ha sido el TLE4208G. Este integrado, fabricado por Infineon Technologies, es un puente en H capaz de proporcionar una corriente de salida de hasta 1A a una tensión máxima de 45V. Incluye diodos de protección por lo que basta con colocar el motor, cuyo sentido de giro deseamos controlar, entre los terminales de salida del integrado. Incluye dos puentes en H, por lo que con un único integrado se pueden controlar los dos motores [4.26].

4.3.3. Tarjeta de red

La tarjeta de red utilizada también pertenece al sistema de desarrollo de Gumstix y es la denominada netCF-vx (ver figura 4.23). Contiene un puerto Ethernet y un zócalo para Compact Flash, que será donde resida la aplicación a ejecutar. En la aplicación desarrollada se ha incluido una memoria Compact Flash de 1 GB de memoria para el almacenamiento de los programas.

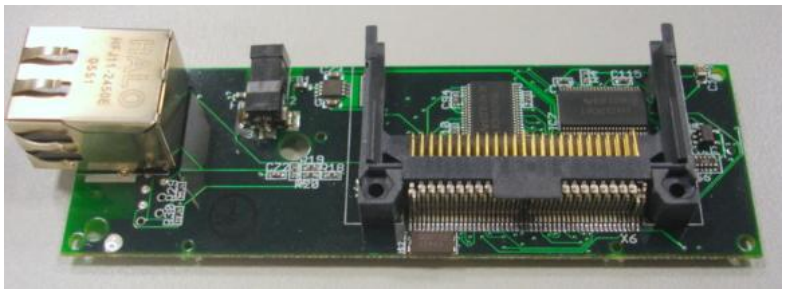


Figura 4.23. Tarjeta de red netCF-cv, de Gumstix

4.3.4. Servidor Sensorial

Como Servidor Sensorial, centro de gestión de sensores y actuadores, de nuevo se ha recurrido a una tarjeta de expansión del sistema Gumstix, en este caso la tarjeta Robostix, figura 4.24.

Esta placa que incluye un micro-controlador de 8 bits Atmel [4.27], el ATmega128 [4.28], junto con el resto de componentes necesarios para su correcto funcionamiento así como conectores para tener acceso a prácticamente todos los pines de entrada/salida de dicho micro-controlador. Es en esta tarjeta donde se conectan los sensores del robot, para que sea posteriormente enviada esa información a la tarjeta

Gumstix que a su vez la enviará al cliente que lo solicite. La conexión de ambas plataformas se realiza por puerto serie.

El elemento fundamental de la placa Robostix es el micro-controlador ATmega128 de la familia AVR de Atmel. Se trata de una familia de micro-controladores de 8 bits con un juego de instrucciones reducido (RISC), arquitectura Harvard y que integran una gran variedad de periféricos. El ATmega128 incluye 128 kBytes de memoria Flash para programas junto con 4 kBytes de memoria SRAM y 4 kBytes de memoria EEPROM. Tiene un juego de 133 instrucciones, la mayoría de las cuales son de un ciclo de ejecución y puede ejecutar 16 millones de instrucciones por segundo. También incorpora dos temporizadores de 8 bits y dos de 16 bits, dos puertos de serie, convertidor analógico a digital de 10 bits de resolución y comparador analógico.

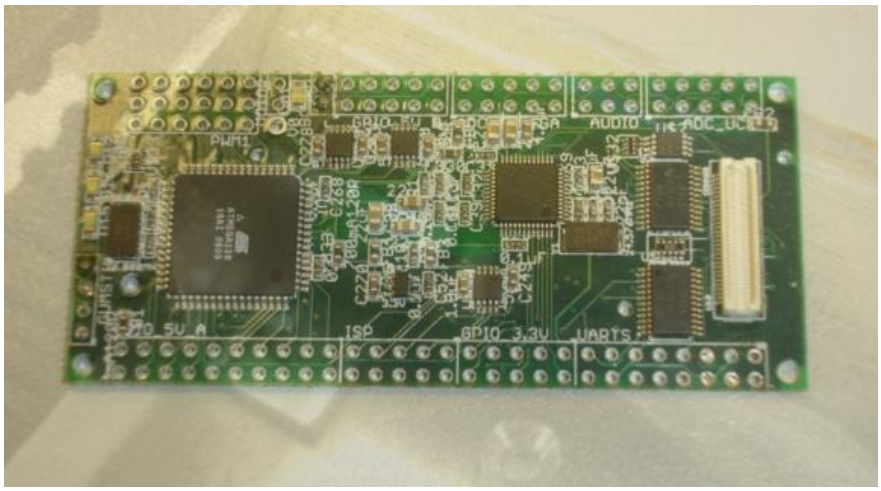


Figura 4.24. Tarjeta Robostix, de Gumstix

Atmel distribuye de manera gratuita un entorno de desarrollo integrado, el AVR Studio [4.29]. Es una herramienta muy completa que incluye ensamblador, compilador de C, simulador y depurador, aunque solo está disponible para Windows. Dado que el desarrollo del proyecto se ha hecho en Linux, en ocasiones se han utilizado herramientas de código abierto, que también se encuentran disponibles.

La grabación de los programas en la memoria Flash se realiza mediante el protocolo ISP (*In-System Programmer*). En la placa ya se encuentra la parte hardware del programador y en el entorno de desarrollo de Gumstix se ha incluido la aplicación *uisp* para grabar los programas.

4.3.5. Sensores de distancia

Se han utilizado los mismos sensores ya descritos en el punto 4.2.5 anterior.

4.3.6. Conjunto completo montado

En la figura 4.25 se muestra el conjunto completo montado basado en el sistema Gumstix, montado en la parte superior. Se distingue la antena de Bluetooh, que se utiliza para comunicarse con el resto de robots y la unidad central, si la hubiera, y la tarjeta Compact Flash donde reside la aplicación. En la parte frontal dispone de un sensor de distancia tipo SFR-05, que utiliza ultrasonidos, así como una cámara. Como se puede apreciar por la referencia del bolígrafo, tiene un reducido tamaño.

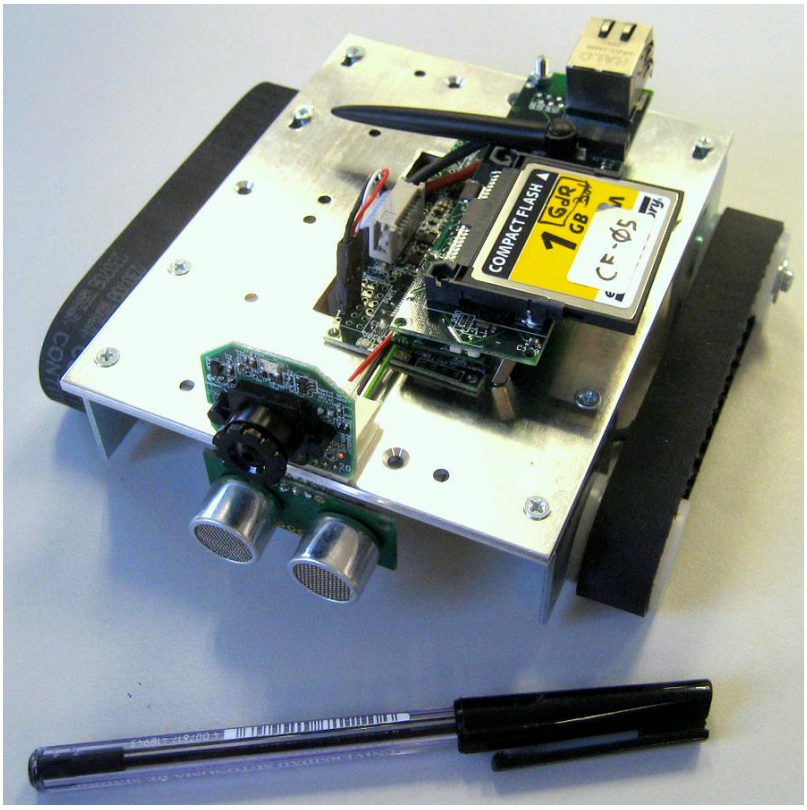


Figura 4.25. Implementación basada en Gumstix.

4.4. Implementación basada en un sistema embebido X-Scale (NSLU2)

El Linksys NSLU2 [4.30], figura 4.26, fue diseñado por la empresa Linksys como un dispositivo para la conexión de discos duros a través de una red. Pero también se puede considerar como un ordenador polivalente basado en un microprocesador PXA.

La comunidad Linux ha lanzado diferentes distribuciones diseñadas específicamente para este dispositivo, como OpenSlug [4.31] o DebianSlug [4.32].

Los principales beneficios de este dispositivo es el precio (~100 €) y que se puede comprar en la mayoría de tiendas de suministros informáticos. Por defecto, cuenta con dos puertos USB y un puerto Ethernet.

La instalación de la distribución de Linux DebianSlug es muy sencilla y se puede obtener en Internet siempre actualizada. Una desventaja de este dispositivo es que sólo tiene puertos USB para ampliarlo o conectar cualquier equipo externo. Para solucionar esto, los Servidores Sensoriales que se conecten a este equipo deben estar provistos de un puerto USB.

Figura 4.26. Plataforma NSLU2, de Linksys

Se puede utilizar en los mismos escenarios que el sistema anterior, el Gumstix, pero es mucho más fácil de obtener, es más económico y dispone de una gran comunidad de desarrolladores que aportan todo tipo de soporte tanto software como hardware. Como contrapartida, es más grande (tiene un área de 108 cm^2 frente a los 16 cm^2 de Gumstix, y menos flexible que la Gumstix, porque sólo se pueden utilizar sensores o actuadores cuya acceso esté basado en USB.

También hay algunos grupos de investigación que utilizan este dispositivo, pero menos que en Gumstix [4.33].

La plataforma que se ha implementado sólo incluye la instalación de todo el software, pero no se ha incluido ningún sistema de tracción ni sensores. Esto ha sido suficiente para validar la plataforma, pudiendo realizar las pruebas y experimentos mostrados en el capítulo 5. Se puede considerar como un nodo o agente estático, en contraposición con las implementaciones descritas en los puntos 4.2 y 4.3.

4.5. Implementación basada en FPGA

Cada día es más común que dentro de las FPGAs se puedan implementar potentes microprocesadores o incluso dispongan de ellos integrados en su interior, siendo adecuados para instalar Linux. También se dispone de herramientas específicas diseñadas para este tema, como el EDK [4.34] de la empresa Xilinx [4.35]. Sin lugar a dudas, esta es la arquitectura más flexible. Todo se puede personalizar. Pero, por otro lado, esto hace que el uso de esta arquitectura sea mucho más difícil que otras.

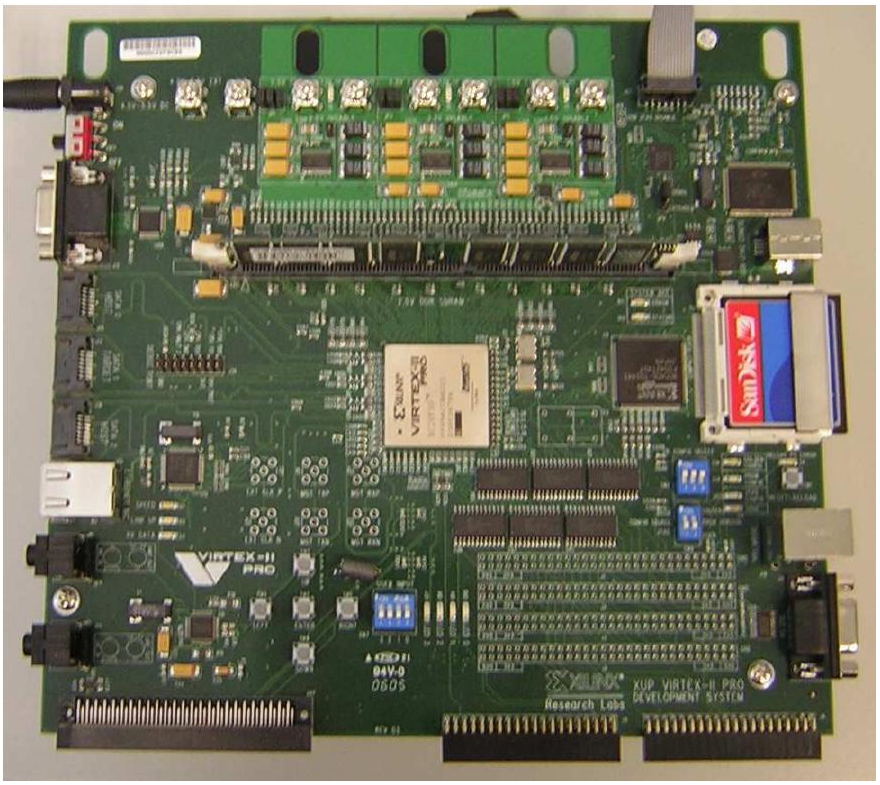


Figura 4.27. Sistema de desarrollo para FPGAs

Para que los proyectos basados en FPGAs sean más asequibles, los fabricantes de estos dispositivos ponen a disposición de los diseñadores sistemas de desarrollo completos, incluidas herramientas software y plantillas para los dispositivos más comunes. En la figura 4.27 se muestra la empleada en este trabajo, basada en una FPGA de Xilinx, modelo Virtex-II Pro.

Para implementar la plataforma GdRBot en este sistema, que sin duda ha sido el más complejo, se han tenido que realizar los siguientes cambios. En primer lugar, la arquitectura tiene que estar definida a partir de las primitivas de las se dispone en las herramientas de diseño de la FPGA. En esta etapa, se pueden definir nuevos sensores en la FPGA, integrando los *drivers* de control correspondientes.

Después de que se han interconectado todos los elementos seleccionados, se debe crear el mapa de memoria. Con los dispositivos y sus buses definidos junto con las conexiones entre esos buses y el mapa de memoria definido se genera el fichero de configuración de la FPGA.

Una vez programada la FPGA, se debe generar un nuevo núcleo de Linux para esta nueva arquitectura. MontaVista [4.36], empresa líder en la comercialización de sistemas Linux para embebidos, proporciona los controladores del *kernel* de Linux para la mayoría de los *cores* más comunes de Xilinx.

Una vez obtenido el *kernel,* se debe desarrollar el resto de la aplicación. Para esta tarea, las herramientas proporcionadas por BusyBox [4.37] son muy útiles para el diseño y desarrollo de las principales herramientas de Linux. Combina pequeñas versiones de muchas utilidades comunes de UNIX/Linux en un único ejecutable pequeño. Proporciona reemplazos minimalistas para la mayoría de las utilidades que se encuentran generalmente en *cores* GNU. Las utilidades en BusyBox generalmente tienen menos opciones que las originales de GNU, sin embargo, las opciones que están incluidas proporcionan la funcionalidad que se espera y se comportan de forma muy similar a sus homólogos de GNU.

Una gran ventaja es que no hay ninguna restricción, aparte de la habilidad del desarrollador, en esta arquitectura no hay nada predefinido. Por otro lado, el desarrollo es mucho más complejo que en los otros sistemas.

También hay que tener en cuenta que los microprocesadores integrados en las FPGAs no son muy potentes. El que dispone el sistema de desarrollo que se ha utilizado está basado en PowerPC 405 [4.38], por lo que no debería considerarse este sistema como un sustituto del PC [4.39]. Actualmente se están integrando procesadores más potentes de tipo RISC de 32 bits.

Finalmente, como aplicaciones recomendadas, se aconseja utilizar esta arquitectura sólo si ninguna de las anteriores se ajusta a sus necesidades. Esta solución es extremadamente flexible, pero también el tiempo necesario para su desarrollo no es comparable con las otras soluciones.

De manera similar al sistema anterior, punto 4.4, con este dispositivo la plataforma que se ha implementado sólo incluye la instalación de todo el software, pero no se ha incluido ningún sistema de tracción ni sensores. De nuevo, esto ha sido suficiente para validar la plataforma, pudiendo realizar las pruebas y experimentos mostrados en el capítulo 5.

4.6. Bibliografía del Capítulo 4

[4.1] "GNU C Library" en www.gnu.org/software/libc/

[4.2] "Building and Testing gcc/glibc cross toolchains", www.kegel.com/crosstool/

[4.3] "The Linux Kernel archives", www.kernel.org

[4.4] "About Apache XML-RPC", ws.apache.org/xmlrpc/

[4.5] "Boa Web Server", www.boa.org

[4.6] "Vía Inc.", www.via.com.tw/en/products/embedded/boards/index.jsp

[4.7] "Debian el sistema operativo universal", www.debian.org

[4.8] “GP_BOT: Plataforma Hardware para la enseñanza de la robótica en la titulación de Ingeniería Informática”. G. Glez. de Rivera, S. López Buedo, I. González, C. Venegas, J. Garrido y E. Boemo, Actas del V Congreso de Tecnologías Aplicadas a la Enseñanza de la Electrónica (TAEE’02), pp: 67-70. Las Palmas de Gran Canaria, Febrero, 2002.

[4.9] "PMD Inc.", www.pmdcorp.com/advanced-motion-control/motion-control-processor.cfm

[4.10] "Módulo de radio de Aurel", www.aurelwireless.com/

[4.11] "Página web de Motorola", www.freescale.com/

[4.12] Página web del micro-controlador MC68HC08GP32: www.freescale.com/webapp/sps/site/prod_summary.jsp?code=HC08G&nodeId=01624684497663

[4.13] "Página web Maxin", www.maxim-ic.com

[4.14] "Página de CodeWarrior Development Tols", www.freescale.com/webapp/sps/site/homepage.jsp?code=CW_HOME

[4.15] "Hoja de datos de L293D", www.ti.com/lit/ds/symlink/l293.pdf

[4.16] "Hoja de datos de CNY70", www.vishay.com/docs/83751/cny70.pdf

[4.17] "Photodetection and Measurement: Maximizing Performance in Optical Systems", M. Johnson (autor). McGraw Hill. ISBN-10: 0071409440. Agosto 2003.

[4.18] "Robot Room: Oscilloscope Traces, Capacitors, and Distance Sensor Results", www.robotroom.com/DistanceSensor3.html

[4.19] "Web de Sharp Microelectronics", www.sharpsma.com/webfm_send/1203

[4.20] "Gumstix Home Page", www.gumstix.com/

[4.21] "Intel XScale Technology", www.intel.com/design/intelxcale

[4.22] “Tiny Linux Computer Has High Hopes For Robotics Applications”, A. Wolfe, InformationWeek, www.informationweek.com/news/164900854, 2005.

[4.23] UltraSwarm: A Further Step Towards a Flock of Miniature Helicopters", R. De Nardi, O. Holland, Swarm Robotics, E. Sahin et al.(Eds.), LNCS 4433, pp 116-128, 2007.

[4.24] "Towards Open Architectures for Mobile Robots: ZeeRO", R.B. Rusu, R. Robotin, G. Lazea, C. Marcu, IEEE International Conference on Automation, Quality and Testing, Robotics, pp.260-265, 2006.

[4.25] "An Autonomous Sailing Robot for Ocean Observation", C. Sauze, M. Neal, TAROS 2006, pp 190-197, September 2006.

[4.26] "Web de Home Infineon Technologies",

www.infineon.com/dgdl/TLE4208G_Green_DS_V1.1.pdf?folderId=db3a30431441fb5d011491e8958a0f76&fileId=db3a30431441fb5d011491ebbe900f79

[4.27] "Web de ATMEL", www.atmel.com

[4.28] "Microcontrolador ATMega128", www.atmel.com/atmel/acrobat/doc2467.pdf

[4.29] "Plataforma de desarrollo AVR Studio", www.atmel.com/avrstudio

[4.30] “Linksys nslu2”, www1.linksys.com/products/product.asp?prid=640.

[4.31] “Openslug”, www.openslug.org/.

[4.32] “Debianslug”, www.slug-firmware.net/d-dls.php.

[4.33] “Twist: A scalable and reconfigurable wireless sensor network tested for indoor deployments”, A. W. Vlado, A. Kopke, *TKN Technical Reports Series*, 2005.

[4.34] “Xilinx embedded design: Platform studio and the edk",

www.xilinx.com/ise/embeddeddesignprod/platformstudio.htm.

[4.35] “Web de Xilinx”, www.xilinx.com/.

[4.36] "Web de MontaVista", www.mvista.com

[4.37] "Web de Busybox", busybox.net/downloads/BusyBox.html

[4.38] “Powerpc 405 embedded cores-IBM Microelectronics”, www06.ibm.com/chips/techlib/techlib.nsf/products/PowerPC405EmbeddedCores

[4.39] “Evaluation of real time performance of embedded linux”, K. C. Truong Quang, V. Vo, Int. Symp. on Electrical-Electronics Engineering, 7 pages, 2005.

Capítulo 5. Resultados experimentales

C.5.

5.1. Introducción

Para comprobar el funcionamiento de la plataforma y su rendimiento en diferentes situaciones, se han efectuado una serie de experimentos con el objetivo de determinar la mejor configuración para cada aplicación.

También se ha utilizado la plataforma para resolver problemas reales y realizar tareas concretas. Esto ha implicado el diseño y fabricación de un nuevo Servidor Sensorial, no previsto inicialmente, lo que ha dado prueba de la flexibilidad de la plataforma y la sencillez para adaptarse a cada situación.

Por último, se han diseñado nuevos Servidores Sensoriales por parte de personas ajenas al presente desarrollo. Sobre éstos se ha implementado el protocolo correspondiente, lo que ha garantizado su correcto funcionamiento al ser conectados a la Unidad de Control.

5.2. Pruebas de diferentes lenguajes de programación y servidores web

5.2.1. Elección del lenguaje de programación

En este primer experimento, se han hecho una serie de llamadas al servidor instalado en el robot desde clientes diferentes instalados también en el propio robot, pero escritos en diferentes lenguajes de programación. Todas las llamadas se hacen a un servidor que ejecuta un CGI de Apache, y dado que dichas llamadas están hechas desde el propio robot los retrasos de red son inapreciables. El experimento se ha hecho sobre la plataforma basada en la placa VIA EPIA 10.000, descrita en el capítulo 4.

Los retrasos que pueden afectar a este experimento son:

- Las demoras para la creación de las tramas TCP/IP, pero como son ejecutadas por el kernel no afectan al resultado pues son constantes en todos los lenguajes.
- Los retrasos debidos a la inserción de una petición XML en el protocolo HTTP, que dependerán de cómo se implementan.
- Los retrasos debidos al *parsing* XML, que también dependen de cómo se implementan, pero son más difíciles de procesar que la gestión del protocolo HTTP.
- Los retrasos debidos al procesamiento de las llamadas.

Los lenguajes elegidos para las pruebas han sido los siguientes:

1.- C usando la librería XML-RPC-c. Está disponible en [5.1], como ejemplo de un lenguaje de propósito general compilado.

2.- PHP utilizando la librería Inutio XML-RPC, disponible en [5.2], como ejemplo de un lenguaje interpretado diseñado para crear páginas web dinámicas.

3.- Java utilizando la librería Apache, disponible en [5.3], como ejemplo de un lenguaje interpretado basado en objetos con precompilación.

4.- C-sharp (c#), plataforma Mono disponible en [5.4] y utilizando la librería de XML-RPC.NET disponible en [5.5], como ejemplo de un lenguaje interpretado basado en objetos y precompilado.

La figura 5.1 muestra los resultados de estos experimentos. Se representa el tiempo entre llamadas para diferentes lenguajes de programación utilizados. Como se puede observar, el lenguaje PHP es más eficiente. Esto se debe a que la biblioteca utilizada para XML-RPC hace que el análisis de XML muy rápido. Además, el transporte HTTP se realiza por una biblioteca central de PHP, que es también muy rápido. Después de PHP, el lenguaje más rápido es C. La razón de que C haya sido más lento que PHP (a pesar de ser un lenguaje compilado) es que la librería utilizada para XML-RPC utiliza unas librerías internas muy lentas para el transporte HTTP [5.6] y para el *parsing* XML [5.7]. El peor resultado es para los clientes escritos en c-sharp y en Java.

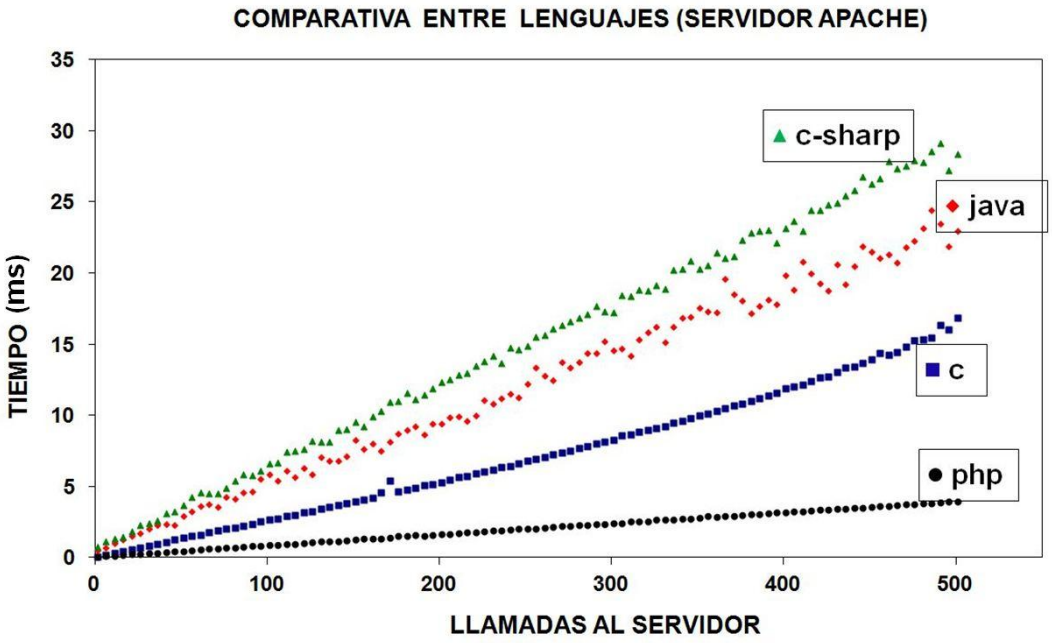


Figura 5.1. Comparativa entre los lenguajes de programación utilizados.

Java muestra cierta dispersión en sus gráficos, lo que se debe a que su máquina virtual es impredecible. C-sharp es el lenguaje de programación más lento y se estima que se debe a que la implementación utilizada (MONO) es todavía demasiado inmadura.

5.2.2. Elección del tipo de servidor web

En este experimento se ponen a prueba dos clientes diferentes (PHP y C) en dos servidores diferentes. Un servidor es un CGI de Apache y el otro es un servidor independiente con el servidor robótico como parte de él. Éste último se basa en el servidor web *Abyss*, disponible en [5.8]. En el gráfico de la figura 5.2, se representa el

tiempo entre llamadas realizado para dos lenguajes de programación y en dos servidores distintos. S observa que C trabaja mucho mejor con el CGI de Apache que con el servidor independiente. En cambio, PHP funcionan mejor en el servidor independiente, pero en un orden inferior. En todos los casos, cada lenguaje tiene un comportamiento lineal.

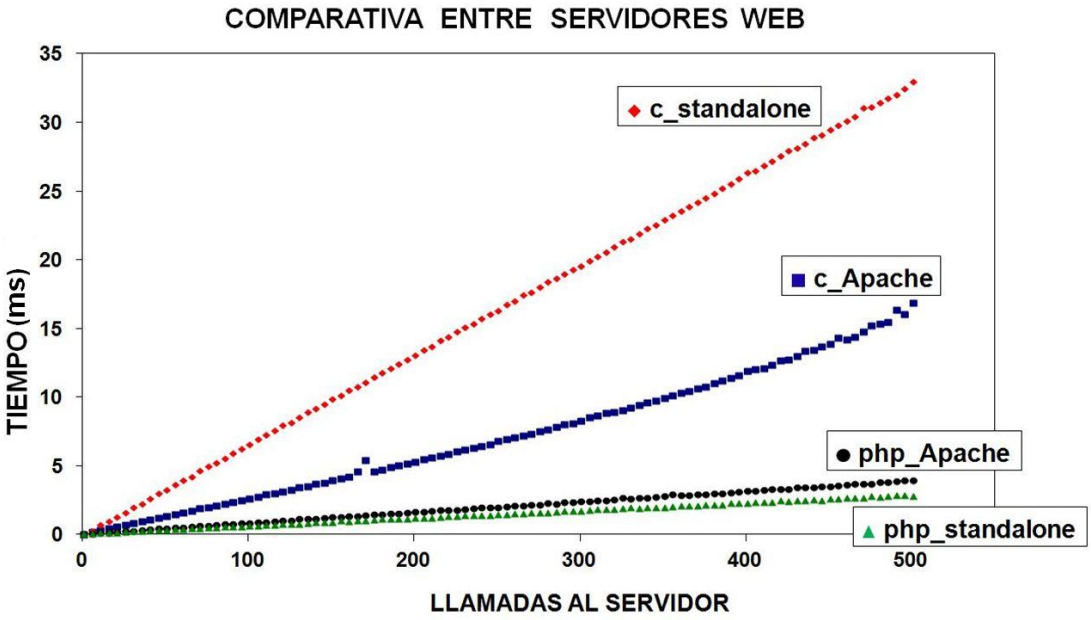


Figura 5.2. Comparativa entre distintos servidores web.

5.2.3. Prueba de llamadas múltiples

Cuando el acceso al servidor es de tipo *polling*, se puede elegir entre dos métodos diferentes. La primera opción es hacer estas llamadas de forma síncrona (antes de hacer la siguiente llamada, se espera la respuesta de la anterior). La segunda opción es hacer las llamadas de forma asíncrona (todas las llamadas se realizan a la vez, sin esperar a las respuestas). En estos experimentos, figura 5.3, se representa el tiempo de llamada de un cliente dentro del servidor y de un cliente externo. Las llamadas se han realizado de forma tanto síncrona como asíncrona. Se han hecho desde dos clientes diferentes, uno en C ejecutándose en el robot y otro en un PC a través de Internet. Ambos se han hecho contra un servidor ejecutándose en el robot como CGI de Apache.

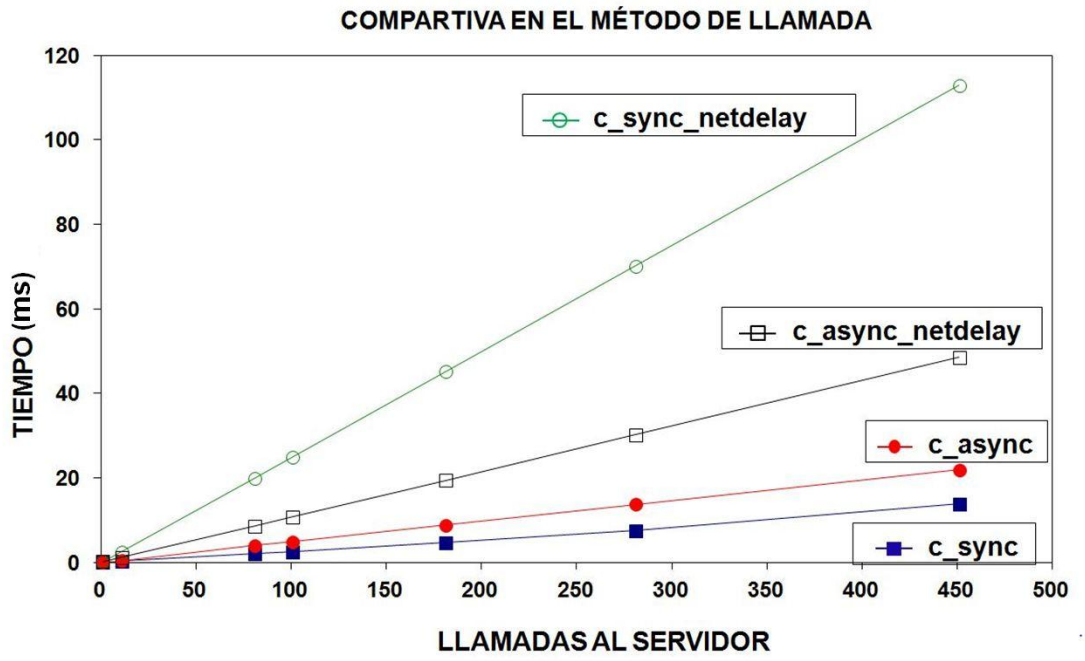


Figura 5.3. Comparación entre métodos de llamada

Si el cliente se encuentra en el robot, el mejor método es el síncrono ya que el método asíncrono produce una sobrecarga y la latencia de la red es cero. Por otro lado, si el cliente se encuentra en un PC en Internet, el mejor método es el asíncrono, ya que la latencia de la red no es despreciable. En el método asíncrono, esto sólo afecta a la latencia una vez, pero en el síncrono afecta a cada una de de las llamadas.

5.3. Pruebas realizadas sobre diferentes plataformas hardware

Se han realizado diferentes experimentos con el fin de comprobar el rendimiento de los diferentes sistemas hardware y configuraciones de software. Todos los experimentos consisten en una serie de llamadas en las que el robot actúa como el servidor, pero con diferentes clientes. Si el cliente es un sistema externo, es una llamada remota, pero si el cliente está en el propio robot es una llamada local. Una llamada típica podría ser la lectura de un puerto de entrada/salida o cambiar un parámetro del motor.

5.3.1. Rendimiento de diferentes implementaciones hardware

En este experimento, se han hecho un conjunto de llamadas a las siguientes plataformas descritas en el capítulo 4: EPIA [5.9], NSLU2 [5.10] y Gumstix [5.11]. Todos

estaban ejecutando un servidor web ligero (Boa) [5.12], debido a que es más apropiado para sistemas embebidos. Las llamadas se han hecho desde un cliente asignado en una máquina diferente (llamada remota).

Como puede verse en la figura 5.4, la plataforma EPIA presenta el mejor rendimiento, como se esperaba, ya que tiene una CPU más potente y puede manejar un mayor número de llamadas por segundo (aprox. EPIA: 180 llamadas/seg, Gumstix: 50 llamadas/seg. y NSLU2: 30 llamadas/seg).

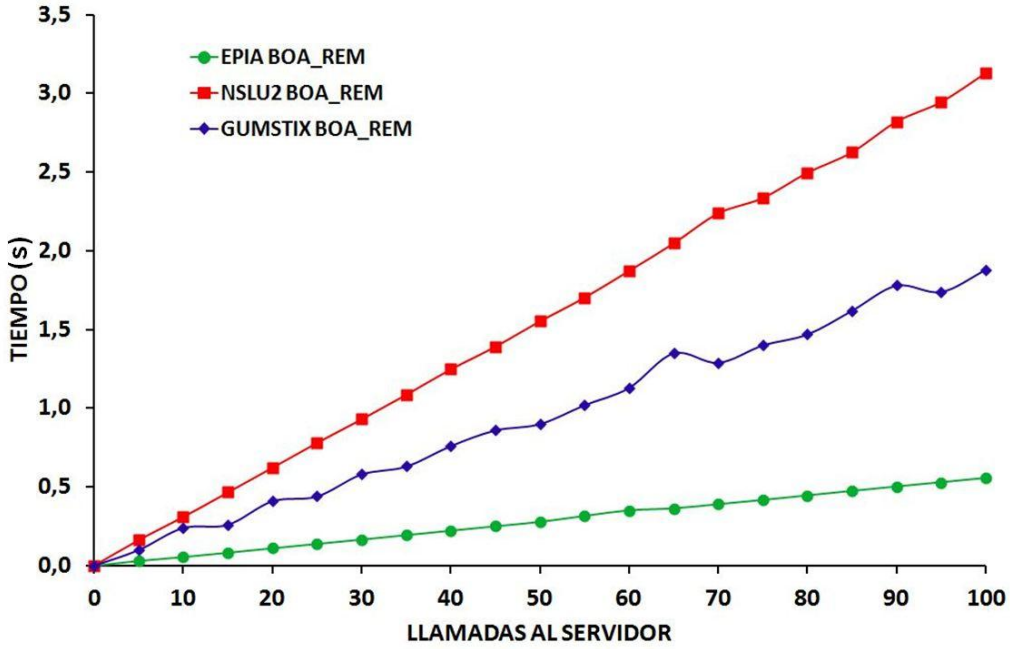


Figura 5.4. Rendimiento de cada arquitectura

5.3.2. Diferentes arquitecturas de servidores web

En este experimento, se han realizado un conjunto de llamadas a un servidor CGI instalado en dos servidores web diferentes, cada uno en una máquina distinta. Como servidores web se han utilizado Apache (más complejo) y Boa (más ligero). Debido a que el servidor web Apache no es soportado de forma nativa por el sistema basado en Gumstix, este experimento sólo ha sido ejecutado en EPIA y NSLU2, ambos conectados a través de Fast Ethernet.

Como muestra la figura 5.5, Boa funciona mucho mejor en ambas arquitecturas. Boa ha sido diseñado pensando en restricciones de velocidad y tamaño. Por otro lado

Apache fue diseñado para proporcionar una gran cantidad de servicios frente al cuidado sobre restricciones de tamaño o velocidad. En el caso de la plataforma GdR_Bot, dado que el conjunto de servicios no es muy grande, es más recomendable el uso de de servidores web ligeros.

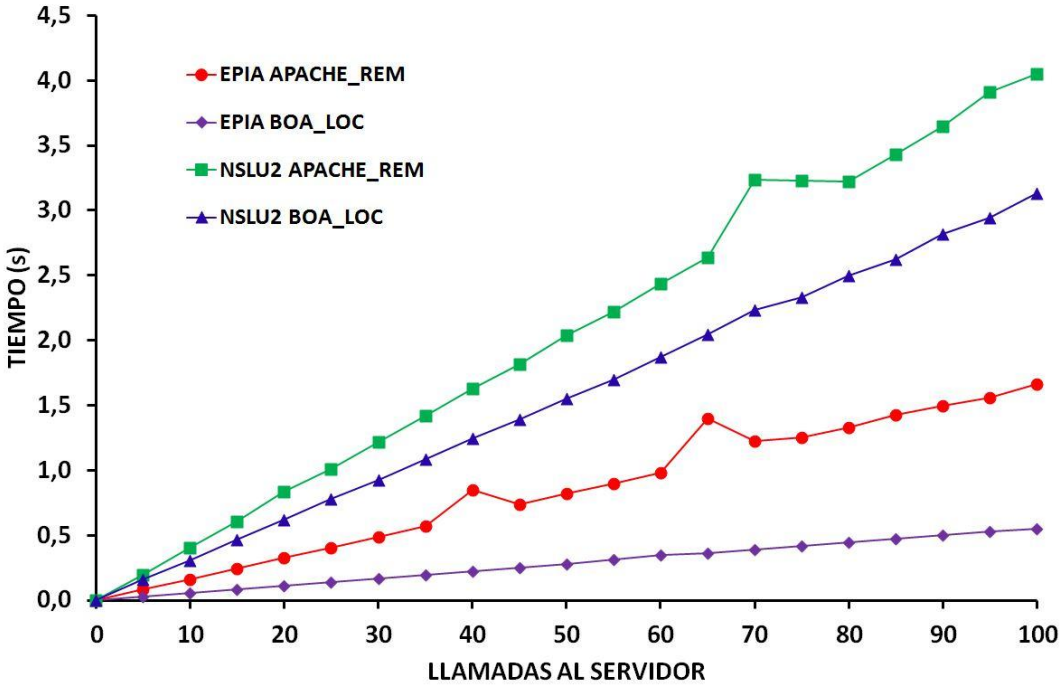


Figura 5.5. Rendimiento de los servidores web

5.3.3. Pruebas con el cliente en diferentes sitios

Para probar la posible importancia de la latencia de la red, en este experimento se han realizado un conjunto de llamadas desde un cliente situado en el robot (local) y desde otro situado fuera (remoto). Este experimento se ha realizado en dos arquitecturas: EPIA y NSLU2, utilizando Boa como servidor web. Como puede verse en la figura 5.6, el procesamiento remoto de la llamada sólo añade una sobrecarga reducida, tanto en la plataforma de EPIA como en la NSLU2.

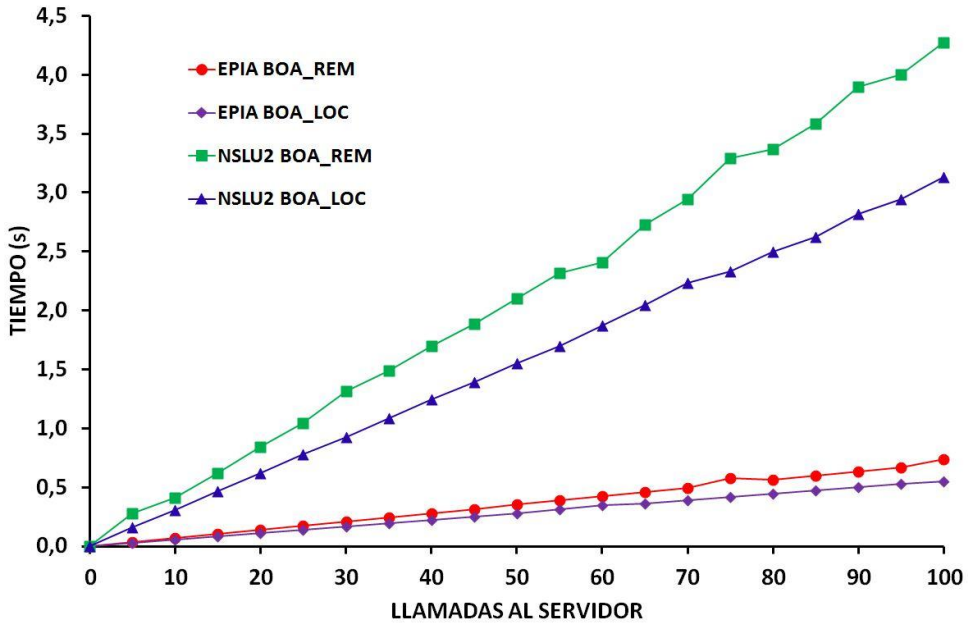


Figura 5.6. Rendimiento por la situación del cliente: local o remoto.

5.3.4. Comparativa de precio versus rendimiento para las diferentes implementaciones hardware

En la figura 5.7 se muestra el rendimiento del sistema normalizado por un lado con el área de la plataforma y por otro con el precio de la plataforma. El rendimiento se ha medido con el servidor web Boa y el cliente remoto.

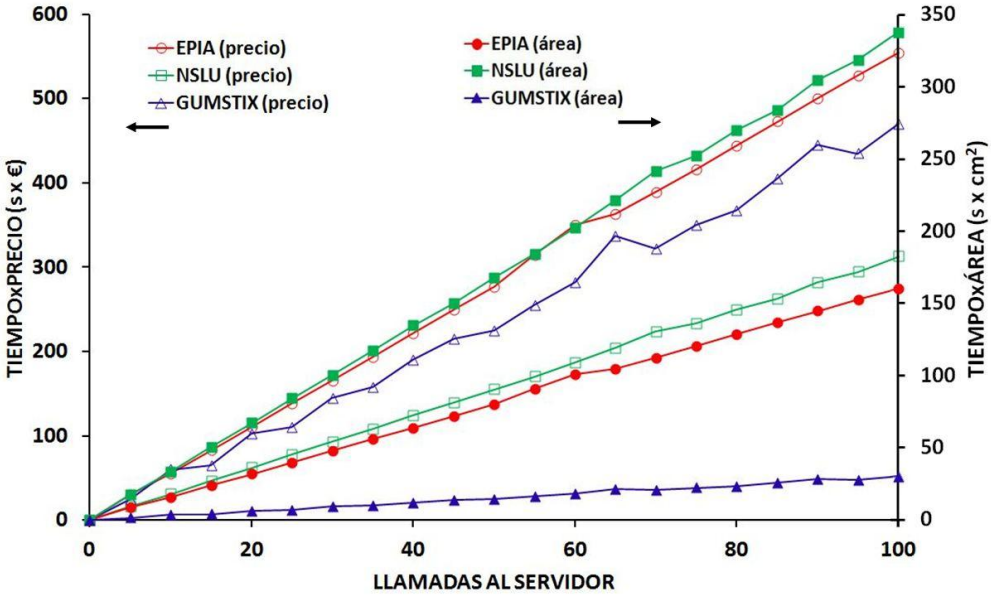


Figura 5.7. Comparativa precio versus rendimiento de cada implementación

Ambos sistemas son muy similares en el rendimiento por precio (puntos abiertos). Sin embargo, la plataforma Gumstix tiene el mejor rendimiento por área (puntos sólidos).

5.4. Experimentos para el procesamiento de imágenes

5.4.1. Configuración del dispositivo

Para la realización de este experimentos se ha utilizado la plataforma basada en la placa VIA EPIA 10.000, descrita en el capítulo anterior, y cámara QuickCam de Logitech webcam de fácil manejo, bajo coste y uso muy extendido.

Figura 5.8. Cámara QuickCam de Logitech

Para el funcionamiento de esta webcam es necesaria la instalación del siguiente módulo del *kernel*, spca5xx. Dicho módulo se puede descargar en la referencia [5.13].

En el caso de las distribuciones de Linux basadas en Debian, se podrá instalar el módulo mediante el siguiente comando:

$~> apt-get install spca5xx-modules

Este módulo creará una interfaz estándar de v4l (*video for linux*) en un dispositivo /dev/videoX (normalmente X = 0, dependiendo de la cantidad de video instalados en el sistema). A través de dicha interfaz las aplicaciones podrán acceder a nuestro dispositivo.

Una vez instalado el módulo es necesario incluir al usuario que va a ejecutar las aplicaciones en el grupo vídeo. Esto se puede realizar mediante un cambio en el fichero de configuración /etc/group. En el caso de que sea el propio Apache, mediante una llamada xml-rpc deberemos realizar una modificación de la siguiente manera (www-data es el usuario por defecto de apache).

video:x:44:www-data

En Linux existen diferentes programas para utilizar la webcam. Estos programas serán de utilidad para comprobar su correcta instalación.

A continuación se comentan los más destacados:

Programas gráficos:

Xawtv: es una aplicación gráfica para acceder a dispositivos de vídeo. Está pensada originalmente para ver la televisión, pero también soporta webcams.

Camorama: Es una aplicación gráfica escrita para Gnome que nos permitirá controlar algunos de los parámetros de la captura de imagen, tales como la luminosidad o el brillo.

Programas en línea de comandos:

Vgrabbj: Es un programa en línea de comandos bastante configurable. Tiene numerosas opciones para la captura y permite exportar fotos a formatos jpeg, png y ppm (más información en la página **man**).

```
$~> vgrabbj –S –F 5 –o png –f imagen.png
```

Webcam: es un sencillo programa pensado originalmente para tomar imágenes periódicamente y subirlas a un servidor remoto ftp. Se configura a través un sencillo fichero de configuración. Un ejemplo de ello se puede encontrar en la página del manual.

Motion: Es una curiosa aplicación para la webcam, la cual está constantemente leyendo del dispositivo, y cuando encuentre un cambio en la imagen (un movimiento) realiza una captura.

5.4.2. Ejemplo de Aplicación gráfica

RoboCam: Para las pruebas de procesamiento gráfico de la plataforma, se ha diseñado y desarrollado un pequeño programa en java para controlar el robot y ver las imágenes obtenidas por la cámara, figura 5.9.

5.4.2.1. Captura de Imágenes

La captura de imágenes a través de la webcam se ha implementado en lenguaje C, se dispone de información al respecto en la documentación del *kernel* (/usr/src/linux/Documentation). También se dispone de código de ejemplo en el código fuente de las aplicaciones mencionadas anteriormente.

Sin embargo esta lectura es bastante complicada porque requiere realizar unos pasos de configuración y se dispone de herramientas como las anteriormente mencionadas que lo hacen correctamente.

Por ello, se ha optado por capturar las imágenes a través de un programa externo. Se ha elegido el programa **webcam** debido a su simplicidad y a la calidad de las imágenes que se obtienen. Además este programa es en línea de comandos y permite utilizarlo sin necesidad de interfaz gráfica.

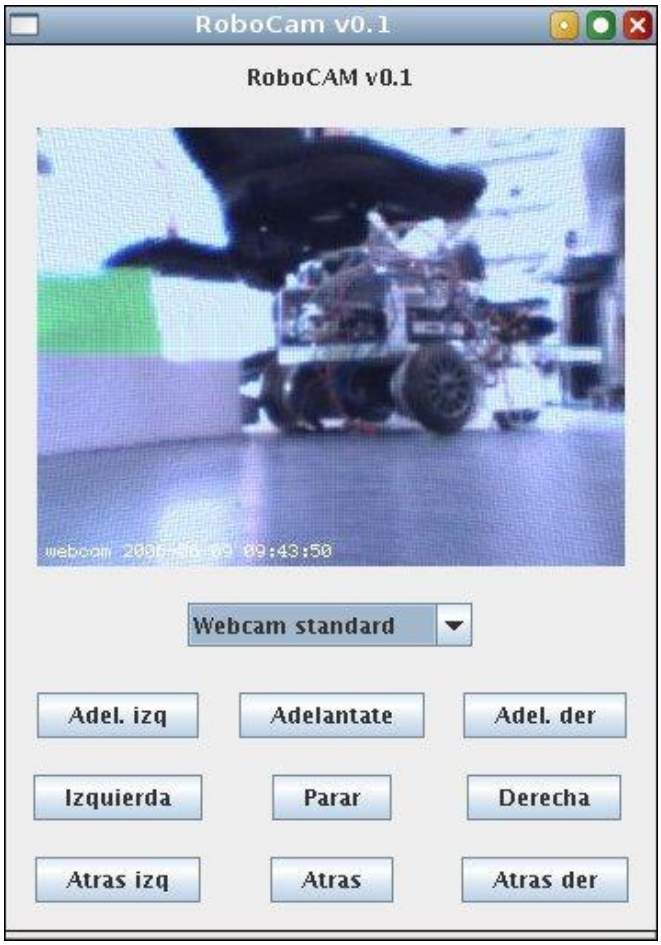


Figura 5.9. Interface de RoboCam

La mayor problemática que tiene capturar imágenes con la webcam es la luminosidad. Dada la escasa calidad del dispositivo, es fácil obtener imágenes muy oscuras. El programa webcam (basado en las librerías de xawtv) hace una corrección automática del color. Si se utilizaran otros programas como xawtv, la mayoría de las veces se debería calibrar manualmente la luminosidad en función de la luz. Se puede observar dicho efecto capturando imágenes con el programa Camorama [5.14]. Cuando se hace

muchas veces se observa que las imágenes tienen poca luminosidad o demasiada, con lo que se debería ajustar mediante controles manuales.

El programa webcam está pensado para capturar imágenes indefinidamente, pero a través de su fichero de configuración se puede especificarle el número de imágenes que se quiere obtener antes de terminar. Se ha observado un comportamiento anómalo, el número de imágenes que se deben capturar debe ser mayor o igual a 2. Si se le especifica capturar una sola imagen, dicha imagen no será correcta. No se ha encontrado la causa de este problema, suponiendo que será del hardware.

5.4.2.2. Detección de colores

En general las imágenes digitales para ordenador están en el espacio de color RGB. La problemática que tiene este espacio de color es que no representa de manera separada características del color, como la tonalidad o el brillo. Es decir, un mismo color tendrá valores muy distintos dependiendo de las condiciones de luz.

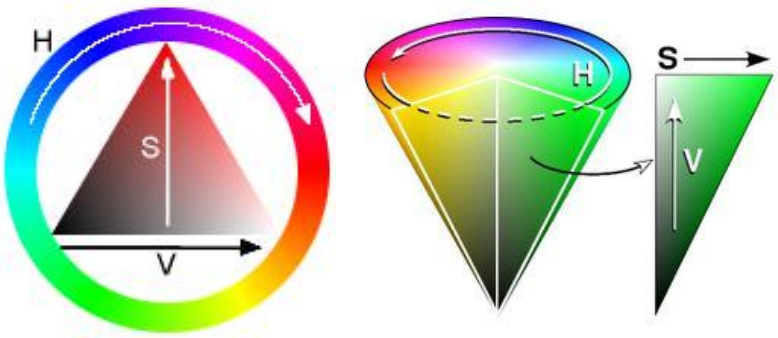


Figura 5.10. Representación del formato HSV

Por ello se ha optado por otro espacio de color: **HSV** (Hue, Saturation, Value). A continuación se define cada componente:

- Hue (tonalidad). Es el tipo de color, tal como rojo, azul, amarillo. Se representa con valores de 0º a 360º, pero también se puede normalizar de 0-100%.
- Saturation (saturación). Es la pureza del color, cuanto menor sea este valor, más gris y decolorado estará. Los valores posibles van de 0-100%.
- Value (valor). Es el brillo del color. Los valores posibles van de 0-100%.

Este espacio de color es muy útil para nuestro objetivo, que es detectar un color en particular, según se muestra en la figura 5.11. Por ello, aunque la imagen inicial está en

formato RGB, se realiza una conversión al espacio de color HSV, para tener en cuenta de manera independiente la componente de la tonalidad (Hue). De esta manera habrá menos problemas con el brillo y la cantidad de luz que capte la webcam.

Para transformar de RGB a HSV, dado un color mediante tres componentes RGB, se pueden aplicar las siguientes fórmulas para obtener su equivalente en formato HSV:

$$H = \begin{cases} 60\ x\ \dfrac{G-B}{MAX-MIN} + 0 & \text{Si MAX = R y G} \geq \text{ B} \\ 60\ x\ \dfrac{G-B}{MAX-MIN} + 360 & \text{Si MAX = R y G} < \text{ B} \\ 60\ x\ \dfrac{B-R}{MAX-MIN} + 120 & \text{Si MAX = G} \\ 60\ x\ \dfrac{R-G}{MAX-MIN} + 240 & \text{Si MAX = B} \end{cases}$$

$$S = \frac{MAX - MIN}{MAX}$$

$$V = MAX$$

Donde MAX y MIN representan las componentes con mayor y menor valor respectivamente.

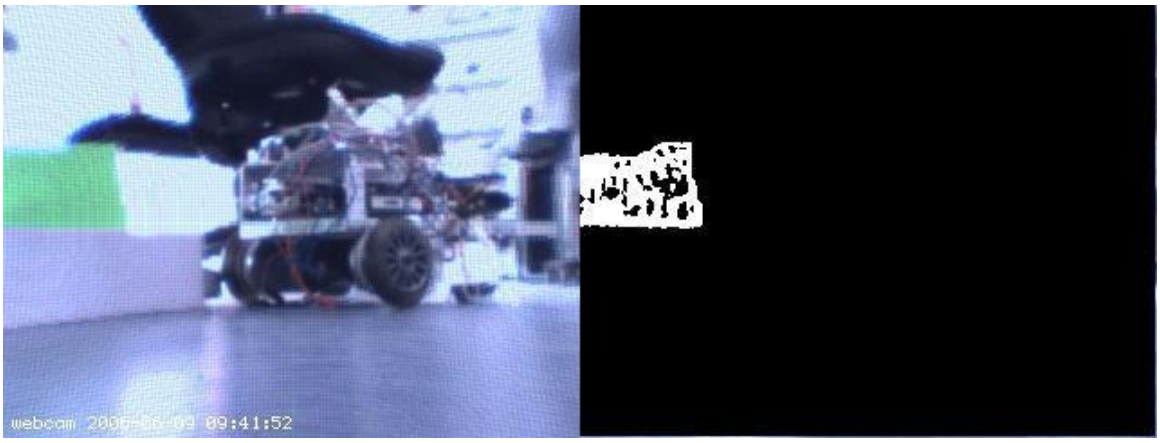


Figura 5.11. Detección de color (en este caso verde)

5.4.2.3. Detección de bordes

Otra de las funcionalidades implementadas es la detección de bordes, figura 5.13, útil para detectar objetos y poder identificarlos de una forma más sencilla y rápida. Para ello se ha utilizado el Operador Sobel [5.15]. Es un operador diferencial discreto que

calcula una aproximación al gradiente de la función de intensidad de una imagen. Para aplicar dicho operador es necesario cambiar la imagen a escala de grises.

El operador se aplica sobre cada píxel de la imagen tomando en cuenta los píxeles circundantes, y el resultado muestra cómo de abrupta o suavemente cambia una imagen en cada punto analizado. Es decir en los píxeles donde cambie de manera brusca la intensidad del color obtendremos un valor más alto (más cercano al blanco), es decir, un borde. Sin embargo obtendremos colores oscuros donde no cambie apenas la imagen.

Matemáticamente se utilizan dos matrices de 3x3 elementos para aplicar la convolución. Se trata de multiplicar cada valor de la matriz por el valor correspondiente en el fragmento de imagen analizado y sumar los valores.

$$Gx = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \qquad \text{y} \qquad Gy = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

Figura 5.12. Matrices de Sobel

Como se puede observar, la primera matriz opera verticalmente y la segunda horizontalmente. El resultado de la convolución será multiplicar cada elemento de la matriz por el píxel correspondiente en el segmento que está siendo tratado. Si hay un cambio brusco de tonalidad, el resultado de la convolución será mayor, dada la disposición de las matrices.

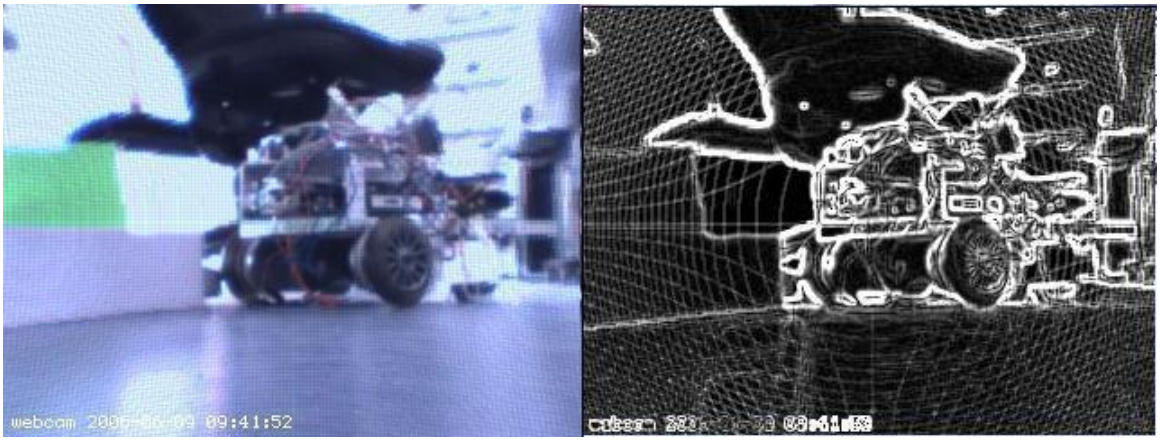


Figura 5.13. Detección de bordes

5.4.2.4. Formatos de imagen

Con independencia de los espacios de colores utilizados, las imágenes han de ser almacenadas en algún formato en el ordenador. Existen numerosos formatos de imagen, pero se han utilizado estos tres:

- JPEG. Es sin duda el formato más extendido, dada su capacidad de compresión y su calidad de imagen. A cambio de una ligera disminución en la calidad de la imagen, se obtiene una imagen mucho más ligera. Por ello es el formato utilizado en la transmisión de datos a través de la red, aunque internamente el servidor opera con otros formatos.
- PPM (Portable PixMap). Este formato de imagen guarda en formato RGB los pixeles de la imagen sin compresión. Lleva una pequeña cabecera inicial que especifica el tipo de imagen, el ancho, el alto y el valor máximo de las componentes. Este formato proporciona una imagen sin pérdida de calidad, pero a cambio, demasiado grande. Sin embargo es mucho más sencilla de tratar. Es la que utiliza internamente el servidor para tratar las imágenes y es el formato utilizado en el programa que busca un color.
- PGM (Portable Greyscale). Similar al anterior, pero en escala de grises, es decir un byte por píxel. Este formato es tan sencillo de leer como el anterior, y además es una tercera parte más pequeño. Utilizamos este formato para la detección de bordes y para el resultado de la búsqueda de un color, ya que solo queremos representar el blanco y el negro.

5.4.3. Aplicaciones

La comunicación de los clientes con el servidor está basada en un modelo cliente/servidor, en el que las aplicaciones cliente realizan peticiones **xml-rpc** al servidor. Xml-rpc es un protocolo de llamada a procedimientos remotos que utiliza xml para codificar las llamadas y http como mecanismo de transporte. Es mucho más sencillo que otros protocolos RPC y además se dispone de librerías para trabajar con él en numerosos lenguajes.

5.4.3.1. Clientes

Aplicación Gráfica Java

La aplicación gráfica nos proporciona una sencilla interfaz gráfica con el robot, ver figura 5.9. A través de ella se pueden pedir imágenes con diversos tratamientos al robot, así como controlar su movimiento. Ha sido implementada en Java 5.0 y consiste básicamente en un hilo que se encarga de actualizar regularmente las imágenes obtenidas por el robot. Estas imágenes son mostradas en una ventana de Swing, junto con los botones de control del robot y el selector de tipo de imagen.

Tanto la actualización de la imagen como las órdenes para el movimiento del robot, se realizan a través de llamadas XML-RPC. A continuación se muestra un pequeño resumen de las clases implementadas y la utilidad de cada una.

- Demonio.java: Esta clase es un hilo que se ejecuta durante todo el tiempo de ejecución de la aplicación. El hilo accede a la ventana para comprobar el tipo de imagen solicitado, y realiza una llamada xml-rpc al servidor, en este caso el robot, con la petición correspondiente. Finalmente actualiza la ventana con la nueva imagen obtenida.
- Motores,java_: Es la clase que se encarga de todos los aspectos relacionados con el movimiento del robot. Cuando pulsamos uno de los botones de la ventana, se realiza la correspondiente llamada “xml-rpc”.
- Ventana.java: Es la ventana gráfica de la aplicación. Contiene los botones, el selector de tipo de imagen y el ImagePanel en el cual se muestran las imágenes del robot.
- ImagePanel.java: Este es el contenedor para las imágenes del robot.
- Main.java: Clase principal. Crea una Ventana con un Demonio corriendo por debajo de ella.

Aplicación de prueba Client

Esta aplicación se encarga de pedir imágenes al robot. Puede ser utilizado como programa de prueba, para probar el correcto funcionamiento. De nuevo se utilizan llamadas xml-rpc, en este caso con la librería de C. Se piden 6 imágenes al servidor y se guardan en archivos para su posterior visualización. Estas imágenes son:

- Imagen normal en formatos ppm y jpeg.
- Imagen con detección de bordes, en formatos pgm y jpeg.
- Imagen con detección de un color, en formatos pgm y jpeg.

Aplicación BuscaColor

Es otro cliente similar al anterior pero más complejo. Escrito en C y basado en llamadas xml-rpc a un servidor. La finalidad es dirigir al robot hacia un color determinado. En este caso el cliente consiste en un bucle de llamadas con el siguiente pseudo-código:

LOOP{

Pide posición del color buscado al robot //xmlrpc

Lógica de decisión

Ordena movimiento al robot.//xmlrpc

}

La lógica de decisión se encarga de decidir el movimiento de los motores en función de la posición del robot respecto al color buscado. Se ha optado por dividir el ancho de la imagen en 5 regiones. En función de la región en la que se encuentre el centro de masas de la imagen obtenida por el robot, los motores giraran a distintas velocidades y sentidos. Por defecto se ha decidido que el robot permanezca parado en caso de que no detecte el color especificado.

Se podría considerar que la lógica de decisión debería formar parte del servidor, pero se ha decidido implementarlo de este modo por una sencilla razón. Puede ejecutar el programa cliente desde cualquier máquina que tenga acceso al robot, pero también podemos ejecutarlo dentro del mismo robot, realizando las llamadas al servidor a través de su interfaz de Loopback. Para ello únicamente hay que cambiar la dirección IP de la constante definida en libmotores.h como URL_ROBOT.

Es posible también ajustar el valor del color que debe buscar el robot, mediante las constantes MAX_COLOR y MIN_COLOR de buscaColor.c, teniendo en cuenta que se refieren a valores en el espacio de color HSV (ver documentación al respecto). Por defecto se ha elegido el color verde ya que es uno de los más sencillos de detectar, y era necesario ajustarse a las limitaciones técnicas de la webcam.

La librería “libmotores.c” se encarga de realizar las llamadas xml-rpc relacionadas con el movimiento de los motores.

5.4.3.2. Servidores

Servidor Webcam

Siguiendo la estructura cliente/servidor, el programa Webcam se encarga de recibir peticiones por parte de los clientes, procesarlas y devolver una respuesta. Es un CGI basado en llamadas a procedimientos remotos, escrito en C y haciendo uso de nuevo de librerías XML-RPC. Por defecto trabaja con una resolución de 320x240.

Los ficheros fuentes de los que se compone el servidor son los siguientes:

- **webcam.c**: fichero principal del servidor xml-rpc, con todos los procedimientos que sirve.
- **libimageutil.c**: librería de tratamiento de imágenes.
- **libjpegutil.c**: librería externa al proyecto para la transformación a formato JPEG.
- **libwebcam.c**: librería para la captura de imágenes de la webcam a través del programa externo Webcams, anteriormente citado.

El servidor registra los métodos que pone a disposición de los clientes, que serán los distintos procedimientos remotos a los que podrá tener acceso. A continuación se detalla el servicio que ofrece cada uno de ellos:

- **GetImage/GetImageJPEG.** Son los servicios que devuelven la imagen sin procesar. Cuando se recibe una petición de este tipo, se captura una imagen y se devuelve en un buffer al cliente, bien en un chorro de bytes en el caso de GetImage (posteriormente el cliente transforma este chorro a PPM añadiendo simplemente la cabecera correspondiente), o bien en formato JPEG para GetImageJPEG.
- **GetImageSobel/GetImageSobelJPEG.** Se diferencia del anterior en que antes de devolver la imagen, la procesan mediante el algoritmo de Sobel para detección de bordes, devolviendo el resultado de dicho proceso a los clientes en formato PGM y JPEG respectivamente.

- **GetImageColor/GetImageColorJPEG.** Estos servicios capturan la imagen, la transforman a formato HSV para poder tratar solo la componente del color y la procesan de modo que cada píxel que esté dentro del rango de color buscado se devuelva como blanco (0xFF) y el resto negro (0x00). Finalmente devuelven la imagen resultante en formato PGM y JPEG.
- **GetImageColorV2.** Este es el servicio utilizado por el cliente buscaColor para obtener la posición relativa del color buscado con respecto al robot. Tras capturar la imagen y durante el recorrido que hace por cada píxel de la misma para pasarla a HSV, realiza el siguiente proceso para obtener el centro de masas del color buscado: Suma las posiciones horizontales de cada píxel que se encuentre dentro del rango del color buscado y la divide por el número total de píxeles que estén dentro del mismo. De este modo se obtiene el centro de masas de ese color para esa imagen (teniendo en cuenta sólo la componente horizontal). Finalmente se devuelve el valor de este punto, que será utilizado por los clientes para decidir el movimiento del robot.

Por último, en la figura 5.14 se muestra la implementación física de la plataforma montada para la realización de las pruebas desarrolladas de captura y análisis de imágenes.

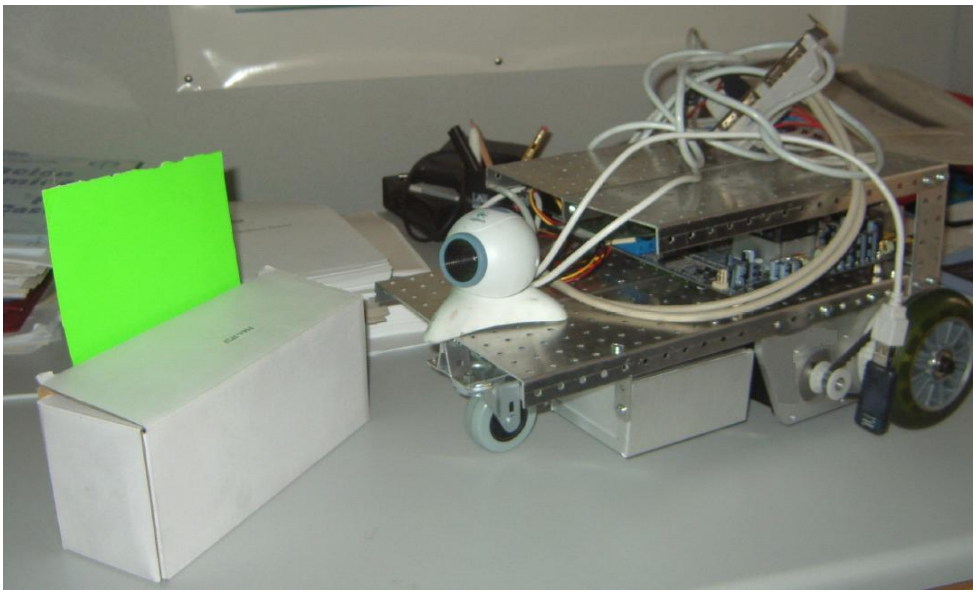


Figura 5.14. Implementación de GdRBot para procesamiento de imagen

5.5. Medidas de intensidad en la comunicación inalámbrica entre robots

En este epígrafe se pretende reflejar a modo de ejemplo, la colaboración que la plataforma GdRBot facilita entre grupos con distintas capacidades y conocimientos.

El proyecto planteado trata de resolver el problema derivado del balizamiento de espacios para la localización de elementos móviles dentro de dicho espacio [5.16] y [5.17]. En estos sistemas de posicionamiento, es condición necesaria conocer con precisión la posición de las balizas emisoras [5.16] o receptoras [5.17], para que por medio de un sencillo algoritmo geométrico de triangulación, poder conocer la posición de los otros elementos fijos o móviles presentes en el área controlada. La necesidad de conocer el posicionamiento de las balizas, supone en ocasiones uno de los inconvenientes en ambas aproximaciones, cuando por ejemplo se desea ampliar el espacio de actividad, sin que exista la posibilidad de ampliar consecuentemente el número de balizas necesario para cubrir el nuevo área.

En colaboración con el grupo del Dr. Anguiano del Departamento de Ingeniería Informática de la EPS, grupo que investiga con Agentes Colaborativos desde una perspectiva software, se ha planteado una nueva aproximación al tema del posicionamiento en interiores.

Se ha desarrollado un algoritmo basado en Agentes para los que, sin ninguna información previa de su posición, pueden auto-localizarse con precisión en entornos desconocidos. Los agentes (software) desarrollado son muy simples y presentan tres funcionalidades claramente identificadas:

1. Facilidad para la transmisión de mensajes.
2. Una memoria de sucesos limitada.
3. Un comportamiento multiestable, variable en función del tiempo y del efecto del último mensaje recibido.

Estas tres funcionalidades permiten diseñar agentes reactivos simples que, en conjunto, tienen un comportamiento complejo. El funcionamiento de cada uno de estos elementos se describe a continuación:

Los mensajes contienen la información que los agentes envían con información exclusivamente de sí mismos. Estos mensajes se envían aleatoriamente y de acuerdo con ciertas reglas según el problema a resolver. Estos contienen, exclusivamente, información acerca del agente que los envía o una retransmisión de la información recibida dependiendo de la naturaleza del problema.

La memoria de sucesos consiste en una pequeña memoria en la que se mantiene parte de la información recibida de otros agentes. La información almacenada será siempre pequeña y se corresponderá siempre con un resumen total o parcial de toda la información recibida.

El comportamiento multiestable consiste en un sistema de tensión-relajación con múltiples rangos de actuación en función de la variable que controla el estado. En la figura 5.15 se muestra de forma esquemática este comportamiento, en donde cada color simboliza un estado y las flechas señalan los valores en los que se cambia de estado. El sistema se relajará siempre en función del tiempo transcurrido y puede tensarse por dos motivos diferentes en función del comportamiento deseado del agente. Los agentes reactivos pasivos se tensarán cuando los mensajes recibidos aporten información incongruente con el estado del agente y los agentes reactivos activos se tensarán cuando la información sea congruente con el estado. Los cambios de estado se realizarán normalmente con un umbral doble. Este umbral doble permite mantener durante más tiempo cada uno de los estados permitiendo un comportamiento más estable que con un umbral único.

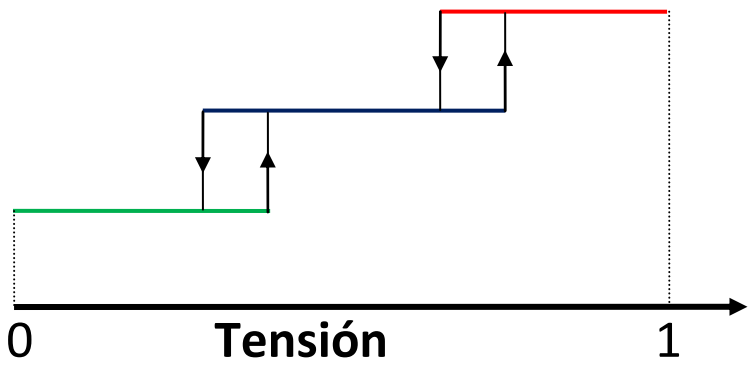


Figura 5.15. Descripción del comportamiento multiestable

Los mensajes que se envían sólo contienen la posición del agente y un identificador del agente emisor. Inicialmente la posición enviada es completamente aleatoria, la

distancia se calcula a partir de la intensidad de potencia con la que se recibe el mensaje. Dado que se está implementando una simulación previa a la implementación física, en la simulación se transmiten también las coordenadas reales con el fin de calcular dicha intensidad (como el inverso del cuadrado de la distancia) con un error en la posición determinado por un parámetro variable, que en este caso será del 10%. Los mensajes se envían con una probabilidad controlada por parámetro con el fin de no saturar la red de comunicaciones. En este caso un agente utiliza sólo un 2% de su tiempo en enviar mensajes (probabilidad 2%). Por último señalar que dicho mensaje se envía aleatoriamente a alguno de los otros agentes.

Se han realizado varias simulaciones del sistema modificando distintos parámetros. A modo de ejemplo señalar que con 30 agentes, tras 700 iteraciones (llamadas simuladas), se ha obtenido un mapa de posicionamiento relativo entre los 30 agentes con un 1% de error con respecto a las posiciones reales. Significar por último que bastaría con fijar dos agentes en posiciones absolutas y conocidas para tener un mapa de posicionamiento absoluto capaz de ubicar a cualquier nuevo elemento, siendo suficiente que éste pueda comunicarse con cualquiera de los anteriores.

Es evidente que el paso de los agentes software para la simulación a los agentes físicos para una implementación real se puede realizar mediante el Sistema GdRBot presentado en esta tesis. En la implementación física de este experimento, cada agente va a tener su propia IP y es necesario implementar algún tipo de sistema de *broadcast* en el que un agente declara su IP al resto. Como puede haber agentes de distintos tipos, la IP haría también la función de identificador del citado tipo.

Para el cálculo de las distancias, es necesario implementar un método para relacionar la información enviada por *wireless* con la intensidad recibida por el mismo medio o cualquier otro. Con el fin de solucionar esto, se ha diseñado y construido un nuevo Servidor Sensorial, descrito en el Anexo II, cuyo fin es la medida de dicha intensidad de potencia entre dos transmisiones de radio. El diseño de este Servidor Sensorial incluye, evidentemente, un desarrollo hardware y, al estar basado en un microprocesador distinto de los Servidores Sensoriales diseñados hasta la fecha, ha habido que diseñar también un nuevo software para el gestor de comandos de forma que cumpla con la interfaz definida. Éste ha sido instalado en una nueva plataforma con la que se está

trabajando desde el año 2009, basado en un microprocesador OMAP3530, figura 5.16, fabricado por la empresa ISEE, modelo IGEP v2 Board. La utilización de esta nueva plataforma sirve como un nuevo ejemplo de la versatilidad del sistema GdRBot, ya que es diferente de los utilizados en el capítulo 4. Las principales características de esta nueva plataforma son:

- TI OMAP3530: ARM CORTEX A8 core + POWERVR SGX 530 core + IVA2.2 + DSP TMS320C64x+
- Gestor de tensiones de alimentación TPS65950
- 4Gb NAND/ 4Gb Mobile Low Power DDR SDRAM @ 200 Mhz
- Ethernet 10/100 Mb BaseT (SMSC LAN9221i)
- Wifi IEEE 802.11b/g (Marvell 86w8686B1)
- Bluetooth 2.0 (CSR BC4ROM/21e)
- Antena integrada y conector para antena externa.
- 1 x USB 2.0 OTG y 1 x USB 2.0 Host
- Conector MicroSD
- DVI-D para conectar un monitor digital.

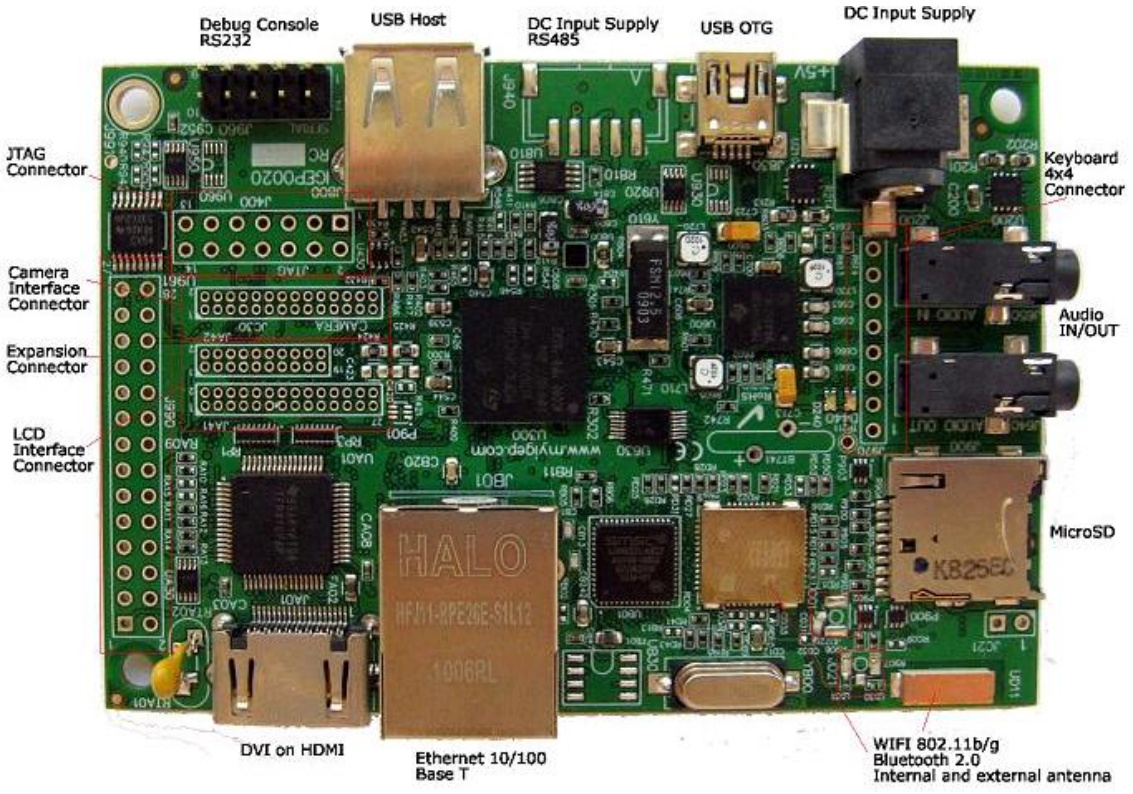


Figura 5.16. Plataforma IGEP v2

Como soporte y sistema de tracción se ha utilizado el diseñado para la plataforma Gumstix, ya descrito en el capítulo 4.

Este mecanismo permitirá tomar las medidas necesarias para realizar la triangulación propuesta en el experimento real que se pretende llevar cabo.

Para comprobar que la calidad del Servidor Sensorial diseñado es la suficiente, es decir el error en el cálculo de la distancia debe ser inferior al 10%, que ha sido el valor simulado, se han efectuado medidas de intensidad de potencia tomadas entre dos agentes físicos, dos robots móviles, implementados bajo la estructura GdRBot con la plataforma descrita en el párrafo anterior. Estas medidas se han tomado con el fin de obtener la relación entre potencia recibida y distancia y el resultado se muestra en las gráficas de la figura 5.17. Sabiendo esta relación, será sencillo calcular la distancia que separa dos agentes conocida la potencia con la que se reciben los mensajes.

En las gráficas se reflejan las medidas realizadas en tres escenarios diferentes, así como una comparación entre todas ellas. Las características y las medidas tomas de cada escenario han sido:

Escenario 1. Un agente fijo (inmóvil) y otro móvil, ambos con un transmisor/receptor. El robot móvil se va alejando en línea recta y toma una medida de potencia cada 25 cm hasta llegar a los 10 metros, en un único eje. Esta situación se repite 10 veces, recorriendo siempre el camino en el mismo sentido y tomando la media para cada distancia.

Escenario 2. En el mismo lugar y los mismos agentes, pero en este caso el robot móvil se va alejando y toma una medida de potencia cada 25 cm hasta llegar a los 10 metros. Entonces regresa por el mismo camino y vuelve a tomar una medida cada vez que retrocede 25 cm, en sentido descendente. Esta situación se repite 10 veces y se toma la media para cada distancia.

Escenario 3. El mismo experimento del escenario 2, pero en otro lugar.

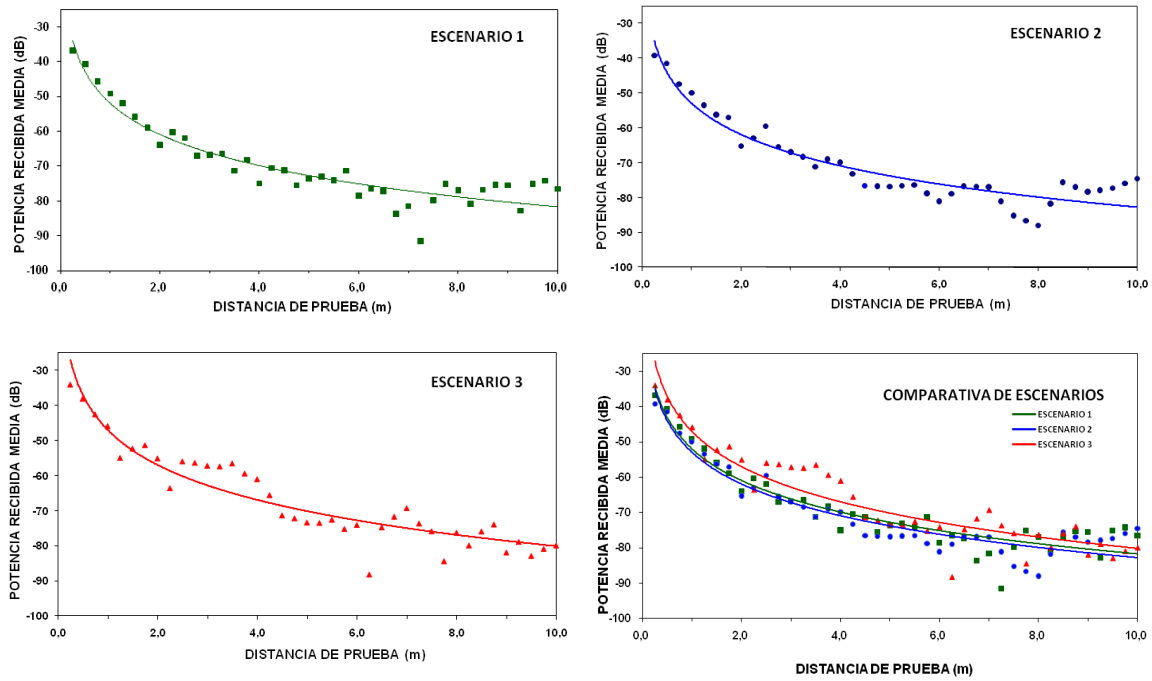


Figura 5.17. Señal en cada escenario y comparativa

En todos los escenarios la propagación de la señal sucede en visión directa entre los módulos de RF (*Line Of Sight*). Los módulos de RF funcionan a una frecuencia de 869,2 MHz, pudiendo hacerlo en la banda de RF ICM Europea de 868,1 a 869,9 MHz con una separación de canales de 100 KHz. La potencia de salida es de 10 dbm (10 mW).

La primera conclusión que se extrae del resultado del experimento es que el Servidor Sensorial ha funcionado perfectamente. Se ha conectado, a través del bus USB, a la Unidad Central de Control y ésta lo ha reconocido, solicitándole los nuevos servicios que puede aportar al sistema. Básicamente, seleccionan una determinado potencia de transmisión y realizan un cálculo de potencia transmitida. Esto servicios se han reportado al cliente, para que pueda añadirlos a su lista de servicios disponibles.

En las gráficas de cada escenario de la figura 5.17, se muestran los valores de potencia medidos para cada distancia, mostrándose la media de todos los valores medidos en cada iteración del experimento. Si bien los puntos parecen algo dispersos, al generar una línea de tendencia de esos puntos se genera una curva logarítmica que decrece con la distancia, tal y como era de esperar pues la potencia recibida decrece con la inversa del cuadrado de la distancia.

En la gráfica de comparativa entre escenarios se ve que todas tienen la misma forma, aunque las líneas de tendencia de cada escenario están un poco desplazadas. Esto es debido a que las medidas dependen mucho del lugar donde se realicen, de los obstáculos, de los espacios más abiertos (ventanas, puertas,…) etc. Se puede extraer la conclusión de que para poder realizar el experimento de localización de agentes con más garantías, menos error, antes debería ser caracterizado el entorno donde se van a hacer las pruebas

5.6. Diseño de Servidores Sensoriales por parte de terceros

A raíz de los trabajos realizados en esta tesis, se propuso un proyecto fin de carrera para desarrollar diferentes Servidores Sensoriales que demostraran que la plataforma se podía ampliar por parte de terceras personas y que la integración de nuevos elementos se podía realizar tal y como se había planificado. Este trabajo ha sido realizado por una tercera persona bajo la supervisión del autor de la tesis y podría haberse incluido como un anexo. Se ha incluido en la memoria al entenderlo como un experimento adicional de la tesis, por tratarse de la prueba práctica, con un usuario inexperto, de uno de los objetivos con los que la tesis se ha desarrollado.

Este trabajo ha sido desarrollado por el entonces estudiante de Ingeniería de Telecomunicación, D. Ismail Rebah Bouaiachi, trabajo que concluyó en diciembre del 2009 con la defensa del proyecto fin de carrera titulado “Plataforma genérica para desarrollo con robots móviles” [5.18]. Señalar que las capacidades previas del estudiante se limitaban únicamente a las asignaturas de la titulación, habiendo cursado como optativa la asignatura de Robótica en el curso precedente.

El objetivo de este PFC es diseñar y construir una serie de módulos, junto con su API correspondiente, que permitan componer un robot de manera sencilla y rápida para su aplicación a diferentes tareas. Al tener diferentes módulos, cada uno dedicado en exclusividad a una tarea, se consigue que cada uno de ellos sea lo más eficaz posible en la función que realiza y por tanto que la unión de todos sea también la más eficiente. El diseño de los módulos se realizó teniendo en cuenta la interfaz de los Servidores Sensoriales descritos en el Sistema GdRBot.

Los módulos que se desarrollaron se muestran a continuación:

1. **Módulo de locomoción:** Resuelve el problema de la movilidad. Se trata de un módulo cuyo objetivo fundamental es dotar a la plataforma de la capacidad de trasladarse de un lugar a otro recorriendo para ello distancias concretas en línea recta, haciendo giros de cualquier ángulo y combinado ambas tareas.
2. **Módulo US&IR:** Resuelve la problemática de la realización de medidas de distancias. Se trata de un módulo cuyo objetivo fundamental es calcular la distancia que hay entre el módulo y el objeto más cercano. Es deseable que tenga la capacidad de realizar medidas mediante el uso de varios fenómenos físicos por si alguno de ellos fuera inviable en el entorno en el que se desarrolla la tarea.
3. **Módulo de Comunicaciones:** Resuelve la problemática de la comunicación de la plataforma con el mundo exterior. Se trata de un módulo cuyo objetivo fundamental es que el robot sea capaz de interactuar bien con el usuario o bien con otras máquinas, tanto enviando información como recibiéndola. Es deseable que no sólo sea capaz de mostrar la información mediante su envío a un PC sino que sea posible la integración de una pantalla para situaciones en las que no sea posible tener un ordenador conectado físicamente a la plataforma.

Una muestra más de la flexibilidad que aporta el Sistema GdRBot ha sido el mecanismo de comunicación que se ha empleado para conectar estos nuevos Servidores Sensoriales a la plataforma. En los desarrollados en la tesis se utilizaron los buses de comunicación serie asíncrono RS-232 y el USB. Pues bien, el elegido para los nuevos Servidores Sensoriales fue el bus serie síncrono I^2C [5.19]. Es un bus de comunicaciones serie, síncrono y bidireccional de dos hilos [5.19], muy usado en la industria, principalmente para comunicar componentes con cierto nivel de inteligencia. La flexibilidad y simplicidad que aporta I^2C hace que sea una alternativa ciertamente interesante para comunicar los Servidores Sensoriales a la Unidad Central de Control.

A continuación se describen cada uno de los módulos que se han desarrollado, información extraída del trabajo fin de carrera [5.18].

5.6.1. Módulo de locomoción

Es un módulo capaz de proporcionar movimiento a la plataforma robótica y trasladarla de un lugar a otro dependiendo de las órdenes recibidas a través de su bus de comunicaciones, figura 5.18.

Las órdenes que es capaz de interpretar el módulo son las siguientes:

- Girar de manera indefinida en cualquier sentido uno o ambos motores
- Girar una distancia determinada (min. 1mm. longitudinal) uno o ambos motores
- Girar a una velocidad concreta (0-100%) uno o ambos motores
- Provocar un movimiento tal que el módulo gire una cantidad de grados determinados, entre 1 y 360, en cualquier sentido
- Conocer si algún motor está girando y en su caso, cuál
- Parar uno o ambos motores
- Saber cuánto han girado uno o ambos motores desde la última orden

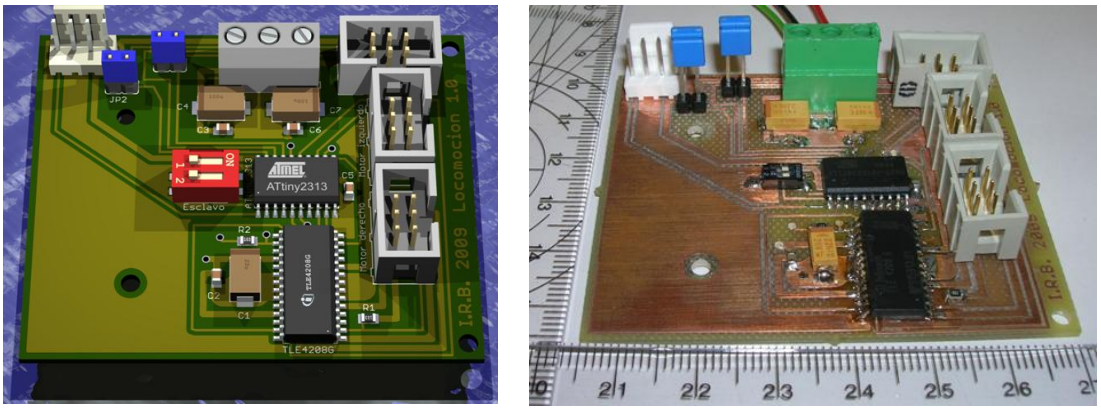


Figura 5.18. Imagen 3D y prototipo de la tarjeta de locomoción [5.18]

5.6.2. Módulo de US&IR

Este módulo es capaz de calcular la distancia a la que se encuentra el obstáculo más cercano al propio módulo. La distancia se calcula mediante sensores de ultrasonidos, mediante sensores de infrarrojos o mediante ambos, dependiendo de las órdenes recibidas, figura 5.19.

Las funciones que puede realizar son:

- Calcular la distancia al obstáculo más cercano mediante uno o varios de los cuatro sensores de ultrasonidos que es posible colocar en el módulo.
- Calcular la distancia al obstáculo más cercano mediante uno o varios de los siete sensores de infrarrojos que es posible colocar en el módulo.

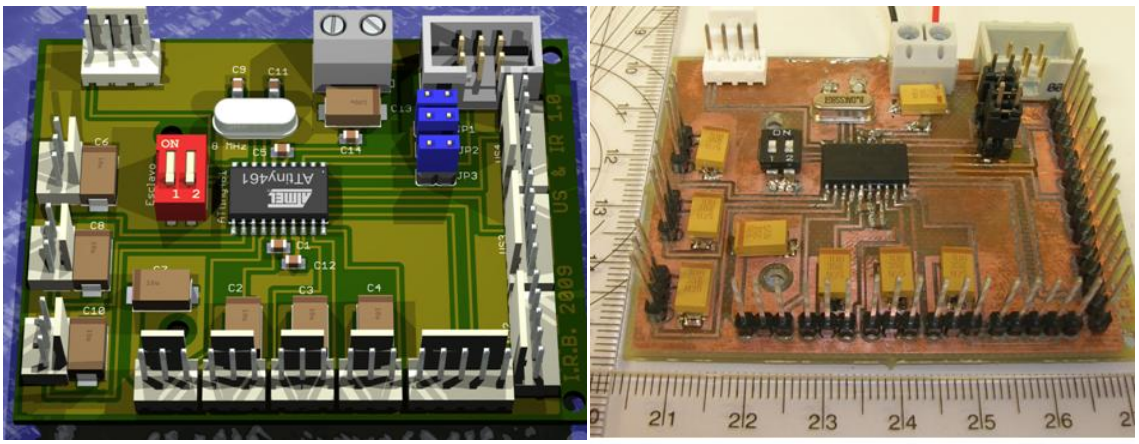


Figura 5.19. Imagen 3D y prototipo de la tarjeta US&IR

5.6.3. Módulo de Comunicaciones

La finalidad de este módulo es que el sistema sea capaz de informar al usuario acerca de algunas situaciones y mostrar datos relativos a los sensores que tiene incorporados con el fin de que haya una interactuación usuario-sistema. La interacción con el usuario puede darse de formas muy diversas y es por ello que se han creado dos modelos diferentes del módulo de comunicaciones, de modo que pueda escogerse el más adecuado a las necesidades de cada aplicación.

En primer lugar, se ha diseñado el modelo A del módulo de comunicaciones. Se trata de un módulo capaz de transmitir y recibir información por los puertos que tiene integrados, que son un puerto USB y un puerto serie RS-232. La información se enviará y/o recibirá por uno de los puertos dependiendo de las órdenes recibidas (Figura 5.20).

En segundo lugar, se ha diseñado el modelo B del módulo de comunicaciones. Se trata de un módulo capaz de realizar las mismas tareas del modelo A pero que además incorpora una pantalla gráfica LCD. Gracias a la existencia de este modelo no sólo se

transmite o recibe información por alguno de los puertos USB o serie de los que está dotado, sino que esa información puede mostrarse por la pantalla LCD junto con imágenes explicativas (Figura 5.21). Las órdenes que es capaz de interpretar este módulo son las mostradas a continuación:

- Recibir y enviar datos por el puerto serie RS-232 (conector DB9 hembra).
- Recibir y enviar datos por el puerto USB (conector USB tipo B).
- Escribir letras y/o números en cualquier lugar de la pantalla LCD, determinado por fila y columna. Se ha desarrollado una tipografía propia.
- Escribir frases y palabras en cualquier lugar de la pantalla LCD con *scroll* automático.
- Mostrar una imagen o un conjunto de ellas por separado o al mismo tiempo en una posición concreta de la pantalla LCD.
- Encender o apagar un pixel concreto de la pantalla LCD.
- Encender o apagar la pantalla LCD para ahorrar energía.
- Borrar todo el contenido de la pantalla LCD sin apagarla.
- Colorear el fondo de la pantalla LCD de cualquier color según una descripción RGB del mismo.

El modelo A, al no disponer de pantalla LCD, sólo admite las dos primeras órdenes.

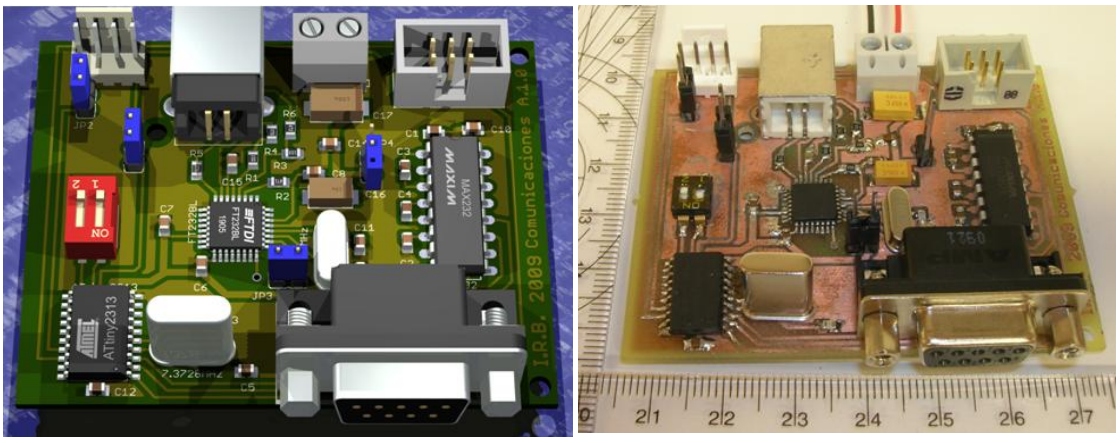


Figura 5.20. Módulo de Comunicaciones, modelo A

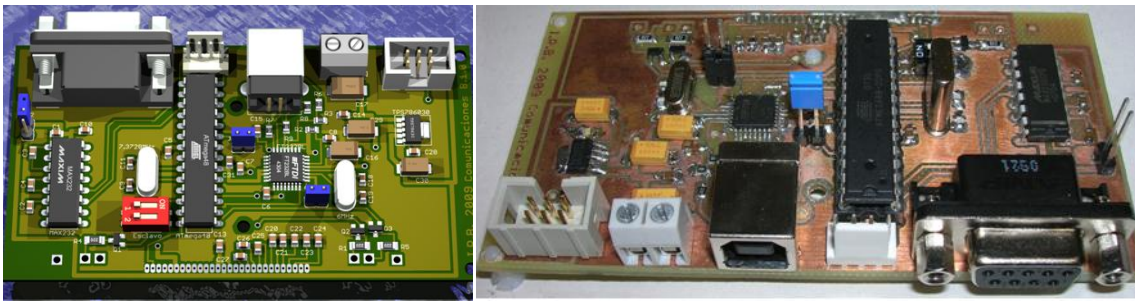


Figura 5.21. Módulo de Comunicaciones, modelo B

5.6.4 Conclusiones

Para probar el funcionamiento de cada uno de estos módulos, en el trabajo de PFC se diseñó una aplicación sencilla, para lo que el proyectando construyó una nueva tarjeta que hacía de Unidad Central de Proceso, muy sencilla, que a su vez le permitía poner un cierre más elaborado a su trabajo. Una vez probado que funcionaban correctamente, se conectaron como Servidores Sensoriales dentro de una plataforma GdRBot ya montada y se comprobó la integración. Las funciones ofrecidas por cada módulo se incorporaron al resto de servicios existentes, se le informó al cliente y éste dispuso de todas para incorporarlas en su aplicación.

Este experimento demuestra la flexibilidad del Sistema GdRBot, la facilidad en añadir nuevas funcionalidades hardware y abre una línea de trabajos futuros en lo que respecta a ir añadiendo nuevos Servidores Sensoriales al Sistema.

5.7. Bibliografía del Capítulo 5

[5.1] "Sourgeforce", sourceforge.net/projects/xmlrpc-c/

[5.2] "Incutio", scripts.incutio.com/xmlrpc/

[5.3] "About Apache XML-RPC", ws.apache.org/xmlrpc/

[5.4] "Mono", www.mono-project.com/

[5.5] "XML-RPC.net", www.xml-rpc.net/

[5.6] "W3C",www.w3.org/Library/

[5.7] "James Clark's Home Page", www.jclark.com/xml/expat.html

[5.8] "ABYSS Web Server", abyss.sourceforge.net/

[5.9] "Vía Inc.", www.via.com.tw/en/products/embedded/boards/index.jsp

[5.10] "Linksys NSLU2", www.linksysbycisco.com/EU/es/products/NSLU2

[5.11] "Gumstix Home Page", www.gumstix.com/

[5.12] "Boa Webserver", www.boa.org/

[5.13] "A free Word LibLand", mxhaard.free.fr/

[5.14] "Camorama Home Page", camorama.fixedgear.org/index.php

[5.15] "Principles of filter design", B. Jähne, H. Scharr, and S. Körkel en Handbook of Computer Vision and Applications. Academic Press, 1999.

[5.16] "The Cricket Project", cricket.csail.mit.edu/

[5.17] "The ActiveBat Project", www.cl.cam.ac.uk/research/dtg/attarchive/bat/

[5.18] “Plataforma genérica para desarrollo con robots móviles”, proyecto fin de carrera de Ismail Rebah Bouaiachi. 5 de diciembre de 2009, Escuela Politécnica Superior, Universidad Autónoma de Madrid.

[5.19] "Página web del bus serie síncrono I2C", www.i2c-bus.org/

Capítulo 6. Conclusiones y trabajo futuro

C.6

6.1. Conclusiones

En esta tesis se ha diseñado e implementado un sistema para controlar de forma sencilla un robot genérico con un conjunto específico de sensores y actuadores conectados. El sistema propuesto permite trabajar sobre entornos inteligentes, centrándose en problemas concretos de la aplicación de usuario, ocultando detalles hardware y de comunicación. Se ha diseñado e implementado un juego de reglas, estructuras y protocolos, denominado GdRBot, que permiten a un usuario no especializado el montaje y control de un conjunto de robots de una forma flexible y sencilla, adaptado a sus necesidades, en el que probar los algoritmos diseñados. Con este sistema se facilita al usuario la capacidad de crear una gran variedad de aplicaciones, permitiendo al mismo tiempo la reutilización de cualquiera de los elementos empleados en soluciones precedentes.

La principal característica del sistema GdRBotr es que no es función ni del software ni del hardware que lo soporta, sólo depende de la especificación de un protocolo.

Dentro de los resultados de esta tesis, se ofrece a los usuarios una serie de elementos de los que disponer para probar sus ideas y son los que componen el Sistema GdRBot. Estos son:

- Elementos software:
    - Servidor Robótico. Programa que actúa como servidor y se ejecuta sobre el sistema operativo Linux. Incluye la interfaz de acceso a todas las funciones que se pueden invocar desde el cliente, a través de XML-RPC.
- Elementos Hardware:
    - Unidad Central de Control. Sistema basado en microprocesador, con los recursos necesarios, tanto de potencia de cálculo como de gestión de periféricos, así como conectividad con el exterior tipo Ethernet o Wifi.

- Servidor Sensorial. Elementos de gestión y control de sensores y actuadores. Incluyen un gestor de comandos, con una interfaz común para todos, que permite el acceso a sus funciones por parte de cualquier unidad de control que conozca dicha interfaz.

El usuario final tan sólo debe decidir, en función de las necesidades de la aplicación y los recursos de los que dispone, qué partes son las que necesita adquirir y cuáles las aporta él mismo.

Con esta decisión se pueden crear tres escenarios diferentes:

Escenario 1. El usuario parte de cero. En este caso, tras documentar las necesidades de la aplicación que pretende desarrollar, deberá adquirir lo siguiente:

- Unidad Central de Control. Además de la arquitectura hardware necesaria, tendrá instalada la distribución de Linux correspondiente y el Servidor Robótico, compilado para esa distribución y la arquitectura del procesador utilizado. Estarán documentadas todas las funciones y servicios implementados en ese servidor. Incluye hardware y software.
- Servidores Sensoriales. Habrá tantos Servidores Sensoriales como necesite para acceder a los sensores y actuadores de los que quiera disponer. Incluye hardware y software así como documentación de la interfaz con la unidad central de proceso.

Escenario 2. El usuario aporta la plataforma donde instalar el Servidor Robótico (un ordenador portátil, una placa de un PC, un sistema de desarrollo utilizado en un proyecto anterior, etc). En este caso lo que deberá adquirir es:

- Aplicación software del Servidor Robótico. Tras instalar Linux en su plataforma, compilará el código fuente del Servidor Robótico para la distribución que haya instalado y para su plataforma.
- Servidores Sensoriales. Análogo al escenario 1.

Escenario 3. El usuario aporta la plataforma y tiene los conocimientos y medios necesarios para desarrollar sus propios servidores sensoriales. Lo que debe adquirir es lo siguiente:

- Aplicación software del Servidor Robótico. Tras instalar Linux en su plataforma, compilará el código fuente del Servidor Robótico para la distribución que haya instalado y para su plataforma.
- Documentación de la interfaz de comunicación entre el servidor sensorial y la unidad de control. De esta forma, programará el gestor de comandos del servidor sensorial para mantener un diálogo correcto.

La arquitectura del sistema está formada por tres tipos de actores: clientes, servidores y elementos de red, bajo el sistema operativo Linux. Los protocolos de comunicación están basados en el estándar XML-RPC sobre TCP/IP.

Para demostrar la funcionalidad y generalidad del sistema GdRBot, se han realizado experimentos sobre cuatro implementaciones distintas de la Unidad Central de Control, equipos unos de gran potencia y otros con recursos más limitados. Se ha realizado un análisis comparativo de velocidad de respuesta en distintas condiciones, cambiando el servidor web, el lenguaje de programación, la ubicación del cliente, etc.

Se han desarrollado diferentes Servidores Sensoriales, basados en arquitecturas distintas. Tal y como estaba previsto, sin más que cumplir con la interfaz definida, todos ellos se han podido añadir a la plataforma de forma transparente. Como ejemplos de aplicación se han utilizado sensores y actuadores tan distintos como sensores de distancia por infrarrojos y ultrasonidos, cámaras de vídeo, transmisores de radio, motores de continua y paso a paso, etc.

Se ha desarrollado un Servidor Sensorial, GP_Bot, que ha pasado a formar parte de las herramientas de desarrollo utilizadas en la asignatura de Robótica Autónoma, impartida en las titulaciones de la Escuela Politécnica Superior de la UAM. GP_Bot está compuesto por tres módulos conectados entre sí, GP_Bot, GP_Bot_Ifaz y GP_Mon.

A través de un Proyecto Fin de Carrera dirigido por el autor de esta tesis, se han realizado nuevos Servidores Sensoriales, basados en micro-controladores de la familia ATMega, de ATMEL, diferentes de los utilizados hasta ese momento, que habían sido de la familia MC68HC08 de Motorola. También se han conectado al sistema aportando nuevas funcionalidades, demostrando la flexibilidad del sistema.

Se ha demostrado la funcionalidad de la plataforma para plantear problemas abordados desde un punto de vista teórico, habiendo realizado con GdRBot una aproximación experimental a un algoritmo de posicionamiento en interiores planteado en un principio desde un punto de vista teórico con agentes software colaborativos. El paso de los agentes software para la simulación a los agentes físicos para una implementación real se ha realizado mediante el Sistema GdRBot, al que se le ha añadido un nuevo Servidor Sensorial. Éste incluye transmisores y receptores de radio para el cálculo de la distancia a través de la medida de la potencia recibida. Cabe destacar que, para este desarrollo, no se ha utilizado ninguna de las plataformas hardware implementadas en la tesis, si no que se ha utilizado una nueva plataforma. Toda la experimentación demuestra la independencia del sistema GdRBot de la plataforma hardware que le soporta. Gracias a la generalidad con la que se han diseñado las especificaciones de la plataforma, se facilita la portabilidad de una determinada aplicación a otra arquitectura diferente. Para el cliente que lo usa también es transparente tanto el hardware como el software que forman el agente robot que quiere controlar.

En el capítulo del estado del arte, se citaba que los retos en investigación están en continuo avance y que no siempre están resueltos y se resumían en una serie de puntos. A continuación se citan esos puntos y para cada uno de ellos se muestra la solución que aporta la plataforma desarrollada en la presente tesis.

- Percepción. *Los agentes físicos deben tener capacidad de usar una percepción de amplio rango, discriminar otros agentes físicos, estimar sus posiciones y la suya propia, entre otras. La percepción es una aplicación básica que se extiende en las aplicaciones robóticas*. En el sistema GdRBot se pueden conectar todo tipo de sensores, determinados sólo por la aplicación a desarrollar. Para ello basta con diseñar el Servidor Sensorial correspondiente al sensor elegido.

- Acciones. *El agente físico ha de ser capaz de controlar su cuerpo físico. En el caso de robots móviles, ha de ser capaz de ejecutar trayectorias, seguir objetivos, los cuales no han de ser necesariamente estáticos*. Se han descrito, al menos, dos mecanismos diferentes de tracción, basados en el control de motores con la precisión suficiente de forma que permiten un desplazamiento preciso y

controlado. Se ha desarrollado una librería completa para un procesador dedicado de trayectorias capaz de manejar dos motores paso a paso.

- Situación y comportamiento. *Aunque las tareas a realizar sean simples, casi infinitas situaciones aparecen debido a los continuados cambios del entorno, como por ejemplo los objetivos móviles, que otros agentes físicos se desplacen en el mismo entorno, etc*. Como ya se ha menciona en el primer punto, se puede añadir cualquier tipo de sensor, pero en el caso particular de la situación y comportamiento del entorno se han implementado dos Servidores Sensoriales. Éstos controlan medidores de distancia basados tanto en ultrasonidos como en infrarrojos, lo que permite hacer un dispositivo tipo sonar, capaz de realizar un mapa de obstáculos del entorno continuamente.

- Tiempo real. *Como la situación cambia continuamente, hay restricciones de tiempo en la toma de nuevas decisiones, re-planificación de trayectorias, y adecuación de los controles*. La Unidad de Control puede ser todo lo potente que se necesite y el sistema operativo instalado garantiza una ejecución en tiempo real, al menos en los términos relativos a la toma de decisiones en robótica móvil.

- Plataforma. *Debe decidirse cuál es la plataforma de experimentación de los agentes físicos que facilite la investigación internacional e interdisciplinaria sobre dichos agentes*. Se han definido las características que debe cumplir la plataforma para que pueda soportar la estructura GdRBot y se han realizado hasta cinco implementaciones diferentes que prueban su correcto funcionamiento.

Queda, por tanto, justificado que todos estos puntos han sido abordados por esta tesis, dando una solución válida y probada para cada uno de ellos.

6.2. Trabajo futuro

Como mejoras futuras en la plataforma, ya se está trabajando en la creación de un sistema de directorio dentro de la plataforma, que permita mostrar los servidores que están activos en cada momento y cuáles son las funcionalidades que ofrecen con el fin de facilitar las tareas de cooperación.

Se ha comenzado a experimentar en la realización de tareas colaborativas entre un número pequeño de robots con el fin de generar mapas de posición y navegación.

Uno de los problemas a resolver en el futuro es la transmisión de grandes cantidades de datos entre sistemas, puesto que XML-RPC no es especialmente eficiente en estos casos. Para esto se están estudiando dos soluciones: comunicación a través de RTP o HTTP sin sobrecarga de XML-RPC. La ventaja de RTP es que soporta múltiples destinos pero la implementación en el servidor y en el cliente es muy compleja. Usando HTTP sin sobrecarga la implementación es más sencilla pero hace un uso poco eficiente de los recursos y con sobrecarga de red responde peor que RTP.

Con HTTP sin sobrecarga se han realizado pruebas satisfactorias, figura 6.1. La prueba realizada fue una cámara en el frente del robot que permitía el guiado por una habitación. Los datos de la cámara eran transmitidos a través de HTTP, de modo que podían ser visualizados desde cualquier navegador web. El robot era controlado a través de un *Canvas* para que el movimiento fuera lo más intuitivo posible para el usuario. La aplicación de control fue desarrollada en C#.

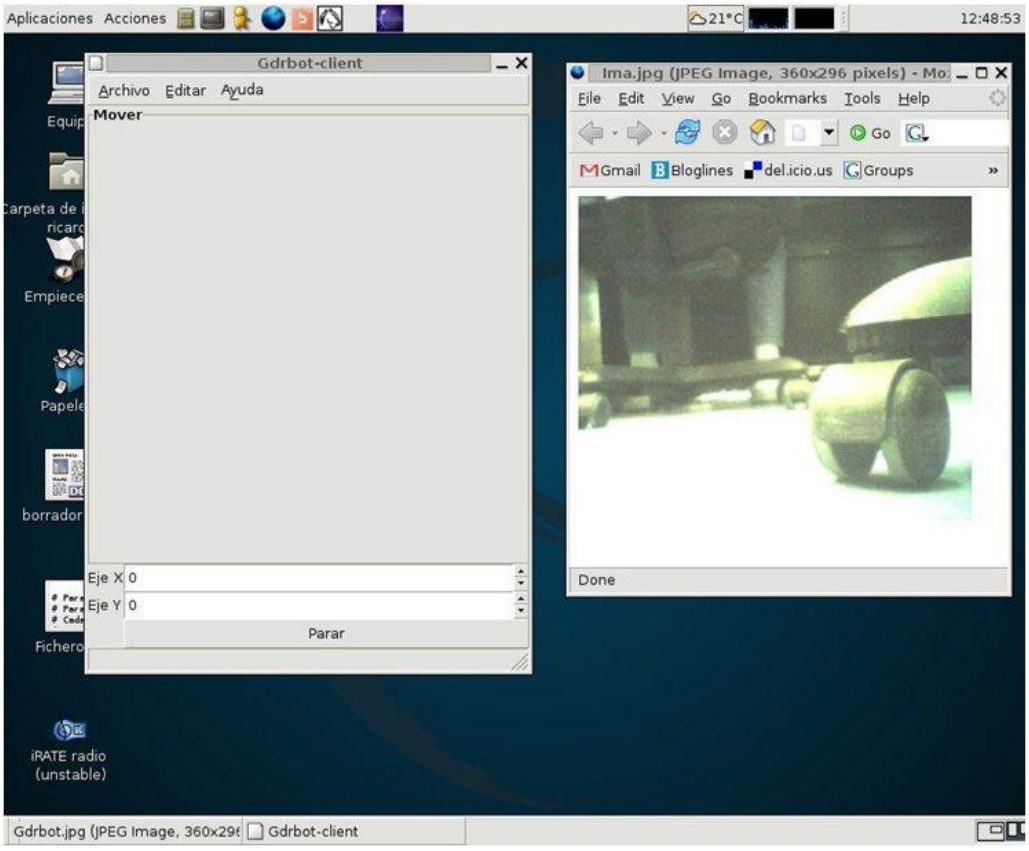


Figura 6.1. Captura de Ejemplo de aplicación de imagen

Así mismo se está estudiando la definición y creación de un sistema de registro global de los elementos que permita la creación de un elemento *Middleware* en el que se puedan redirigir peticiones al agente/sensor más adecuado en función de su disponibilidad/funcionalidad. Con esto se podrían realizar llamadas a la plataforma del estilo: “obtener la temperatura” o “cerrar la puerta” y el sistema buscaría el elemento más adecuado al que realizar la petición.

6.3. La tesis en cifras

A continuación se muestra una lista que enumera los elementos que se han realizado en la tesis, pretendiendo dar una idea del trabajo desarrollado.

6.3.1 Desarrollos software.

Aplicaciones:

- Servidor robótico:
    - Servidor Robótico sólo XML-RPC (standalone).
    - Servidor Robótico que incluye servidor web.
- Clientes
    - Desarrollado en PHP.
    - Desarrollado en perl.
    - Desarrollado en C.
- Servidor Sensorial:
    - Basado en GP_Bot.
    - Basado en GumStix.
    - Desarrollado en el PFC
- Librería en C para Linux el driver de motor MC4310, de Pilot

Lenguajes de programación utilizados:

- Ansi C: 3.153 líneas de código más otras 3.749 en los módulos del Servidor Sensorial desarrollado en el PFC.
- php: 750 líneas de código
- c sharp: 209 líneas de código
- asm: 62 líneas de código
- java: 307 líneas de código
- perl: 130 líneas de código

En el Anexo IV se han incluido todos los ficheros generados a lo largo de la tesis para el desarrollo del servidor, los clientes, ejemplos de aplicación, *driver* para en controlador de motores, etc. Igualmente se listan los ficheros generados para el control de los Servidores Sensoriales desarrollados en el PFC descrito en [5.18].

6.3.2 Desarrollos hardware.

Se han diseñado, construido y montado las siguientes placas de circuito impreso:

- Sistema GP_Bot. Conjunto de tres tarjetas, 20 unidades de cada una.
- Sistema Sensorial RF-USB. Una tarjeta, 10 unidades.
- Sistema Sensorial del PFC. Conjunto de 4 tarjetas, 4 unidades de cada una.
- Driver para motores paso a paso. Una tarjeta, 5 unidades.

Se han utilizado e integrado las siguientes tecnologías:

- Desarrollos con microprocesadores:
    - Procesador INTEL Pentium IV, sobre un ordenador portátil.
    - Procesador VIA EDEN^{TM} ESP, sobre la placa base EPIA TC10.000.
    - Procesador INTEL XScale PXA255, sobre el sistema GumStix y el NSLU2 de Linksys.
    - Procesador PowerPC405, integrado en la FPGA Virtex II Pro.

- Procesador de trayectorias y control de motores *Pilot Motion Processor* MC3410, de PDM.

- Desarrollos con Microcontroladores:
  - Motorola MC68HC908GP32, GP_Bot.
  - ATMEL ATMega128, RoboStix ,de GumStix.
  - ATMEL ATtiny461, módulo US&IR (PFC).
  - ATMEL ATtiny2313, módulo de locomoción y de comunicaciones tipo A (PFC).
  - ATMEL ATMega48, módulo de comunicaciones tipo B (PFC).
  - ATMEL ATMega68, módulo RF-USB.

6.4. Publicaciones

Como resultado de las tareas desarrolladas directamente en esta tesis, se han publicado los siguientes trabajos en el ámbito científico y docente.

"GP_BOT: Plataforma Hardware para la enseñanza de la robótica en la titulación de Ingeniería Informática".
G. Glez. de Rivera, S. López Buedo, I. González, C. Venegas, J. Garrido y E. Boemo.
Actas del V Congreso de Tecnologías Aplicadas a la Enseñanza de la Electrónica (TAEE'02), pp: 67-70. Las Palmas de Gran Canaria, Febrero, 2002. *Premio a la mejor herramienta de laboratorio.*

"A Generic Software Platform for Controlling Collaborative Robotic System using XML-RPC".
G. Glez. de Rivera, R. Ribalda, J. Colás y J. Garrido.
Proc. of IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics (AIM'05), pp: 1336-41. IEEE CAT. nº: 05TH8801C. ISBN 0780390474. Monterrey (USA), Julio/2005.

"Plataforma genérica para desarrollos basados en agentes móviles".
G. Glez. de Rivera, R. Ribalda y J. Garrido.
Actas de Ubiquitius Computing & Ambient Intelligence (UCAmI-05). ISBN: 846096891X, pp: 363-369. Granada, Septiembre/2005.

"Hardware Independent Architecture for Autonomous Colaborative Agents"
G. Glez. de Rivera, R. Ribalda, K, Koroutchev, J.Colas y J.Garrido.
Proc. of 2nd Int. Conf. on Informatics in Control, Automation & Robotics (ICINCO'05), pp: 459-462. Barcelona, Septiembre/2005

"Diseño de un Laboratorio de Robótica Autónoma".
G. Glez. de Rivera, R. Ribalda y J. Garrido.
Actas del VII Congreso de Tecnologías Aplicadas a la Enseñanza de la Electrónica. (TAEE'06). Madrid. Julio 2006.

"Plataforma para el diseño de "drivers" software de alto y bajo nivel para dispositivos hardware".
G. Glez. de Rivera, R. Ribalda, E. Anguiano y J. Garrido.
Actas del VII Congreso de Tecnologías Aplicadas a la Enseñanza de la Electrónica. (TAEE'06). Madrid. Julio 2006.

"A Remote Control Platform for Collaborative Robots".
G. Glez. de Rivera, R. Ribalda y J.Garrido.
Proc. of the 12th Int. Conf. on Robotics and Applications (RA'06), pp: 108-113. ISBN 0889865957, Honolulu (USA). Agosto/2006.

“Performance of an open multi-agent remote sensing architecture based on XML-RPC in low-profile Embedded Systems”
G. Glez. de Rivera, R. Ribalda, A. de Castro y J. Garrido.
Advances in Intelligent and Soft Computing, Vol 55, DOI: 10.1007/978-3-642-00487-2_55 pp 520-528. ISBN: 97836420048672. Springer. Proc. of 7th Int. Conf. on Practical Applications of Agents and Multi-Agent Systems (PAAMS’09), Salamanca. Marzo/2009.

Otra serie de trabajos realizados en el grupo de investigación al que pertenece el doctorando y que han utilizado conceptos previamente desarrollados durante la realización de este trabajo de tesis son:

"Low Cost Indoor Ultrasonic Positioning Implemented in FPGA"
A. Sánchez, S. Elvira, A. de Castro, **G. Glez. de Rivera**, R. Ribalda y J. Garrido.
Proc. of the IEEE Industrial Electronics Conference (IECON'09), pp: 2709-2714, Porto (Portugal), noviembre 2009.

"FPGA-based Embedded System for Ultrasonic Positioning"
A. Sánchez, A. de Castro, **G. Glez. de Rivera** y J. Garrido
Proc. of the IEEE Int.al Symp. on Industrial Electronics (ISIE'10), pp: 3051-3056, Bari (Italy), julio 2010.

"Diseño de una mano mecánica para deletreo dactilológico en LSE"
V. Vaquero, A. Sánchez, **G. González de Rivera** y F. López-Colino,
Proc. of INTERACCION, vol. 1, nº. 1,pp. 471-472, Valencia (España), septiembre 2010.

"Robotic Platform for Localization Based on Ultrasound-Encoder Sensor Fusion"
I. Cortes, A. Sánchez, S. Elvira, **G. Glez. de Rivera**, A. De Castro y J. Garrido
Proc. of Int. Conf. on Design of Circuits and Integrated Systems (DCIS'11), Albufeira (Portugal), noviembre 2011.

"Design of a Robotic Hand Applied to the Representation of the Spanish Sign Language Fingerspelling Dictionary"
V. Vaquero, F. López-Colino, **G. Glez. De Rivera** y J. Garrido
Proc. of Int. Conf. on Design of Circuits and Integrated Systems (DCIS'11), Albufeira (Portugal), noviembre 2011.

Anexo I. Manual de usuario de GP_Bot

A.I.

A.I.1. Introducción

Este sistema de desarrollo, planificado como Servidor Sensorial, surge de la necesidad de contar con un sistema flexible, de propósito general y de cierta potencia para el control y gestión de diferentes sensores y actuadores de una manera autónoma, para ser llamado desde un gestor central. También se puede utilizar como sistema independiente para aplicaciones sencillas.

Un buen sistema de desarrollo debe contar, por un lado, con los recursos suficientes para el control de los elementos básicos que puede necesitar lo que denominaríamos como un robot básico, tales como motores, sensores de infrarrojos, pulsadores, etc. Y por otro lado debe tener un mínimo de capacidad de proceso así como mecanismos de comunicación para su conexión a otros equipos.

La solución que se ha adoptado utiliza un micro-controlador de Motorola, con bastantes herramientas de apoyo y completa documentación. Esto será suficiente para la mayoría de las aplicaciones.

También se ha incluido un módulo de radio que permite una comunicación bidireccional con una estación base, que podrá ser un PC. De esta forma se aumentan los posibilidades de conexión.

AI.2. Tarjeta GP_Bot

AI.2.1. Características de la tarjeta

La tarjeta GP_Bot es una tarjeta de desarrollo de propósito general para el diseño e implementación de sistemas basados en micro-controlador. Está basada en el micro-controlador de 8 bits MC68HC908GP32 de Motorola, cuyas principales características son:

- Arquitectura de alto rendimiento M68HC08 optimizada para compiladores C
- Frecuencia interna del bus de hasta 8-MHz
- Código de seguridad para la lectura y programación de la memoria FLASH
- *Firmware* para la programación desde PC *“on chip*”
- Sistemas de protección: "*Watch Dog*", detección de baja tensión, de direccionamiento ilegal y de código ilegal, todos con *reset* opcional

- Diseño de bajo consumo, completamente estático y varios modos de operación
- 32 Kbytes de memoria FLASH programable en circuito
- 512 bytes de memoria RAM
- Módulo de interfaz serie asíncrono (SPI)
- Módulo de interfaz serie síncrono (SCI)
- Dos temporizadores de 2 canales de 16 bits (TIM1 y TIM2) con captura de entrada seleccionable, comparadores y capacidad de PWM en cada canal
- 8 canales de 8 bits para conversión AD por aproximaciones sucesivas
- Hasta 33 pines de entradas/salidas de propósito general
- Puerto de 8-bits para manejo de teclado
- 16 modos de direccionamiento
- Instrucciones optimizadas de multiplicación (8x8) y división (16x8)
- Optimización para aplicaciones de control
- Soporte eficiente del lenguaje C

Este microcontrolador dispone de un modo especial de trabajo, denominado Modo Monitor, a través del cual se puede acceder desde un PC a todos los recursos internos, acceso a los registros, a los contenidos de la memoria, tanto ROM (flash) como RAM, se puede programar la FLASH interna, permite la ejecución pasa a paso del programa almacenado, etc.

Las características de la tarjeta son:

- Microcontrolador MC68HC908GP32
- Acceso externo a todos los pines del microcontrolador, lo que incluye:
  - Puerto A: 8 bits de entrada/salida digital u 8 bits para la conexión de un teclado
  - Puerto B: 8 bits de entrada/salida digital u 8 canales al conversor Analógico-Digital
  - Puerto C: 7 bits de entrada/salida digital
  - Puerto D: 8 bits de entrada/salida digital o acceso a temporizadores y puerto serie síncrono
  - Puerto E: 2 bits de entrada/salida digital o puerto serie asíncrono
  - Señales de *reset* y entrada de interrupción
- Pulsadores de *Reset* y *Power-On* (arranque en frío*)*

- *Driver* serie tipo RS-232
- Módulo para transmisión-recepción por radio, en la banda de los 433 MHz.
- Sistema de reloj con cristal de 9.8304 MHz
- Fuente de alimentación regulada de 5v
- Entrada de cable monitor para la conexión desde PC, con hardware compatible con Clase I (control de la alimentación a través de la señal DTR del puerto serie)
- Driver serie tipo RS-485 full duplex, opcional (sólo en la versión 1.1)

Para entrar en el Modo Monitor se deben conectar ciertas líneas del microcontrolador a ciertos niveles de tensión, según se especifica en la hoja de datos del microcontrolador, lo que se consigue mediante el uso del denominado Cable Monitor, que incluye un circuito de adaptación de señales y cuyo uso será explicado más adelante.

AI.2.2.Diagrama de bloques de la tarjeta

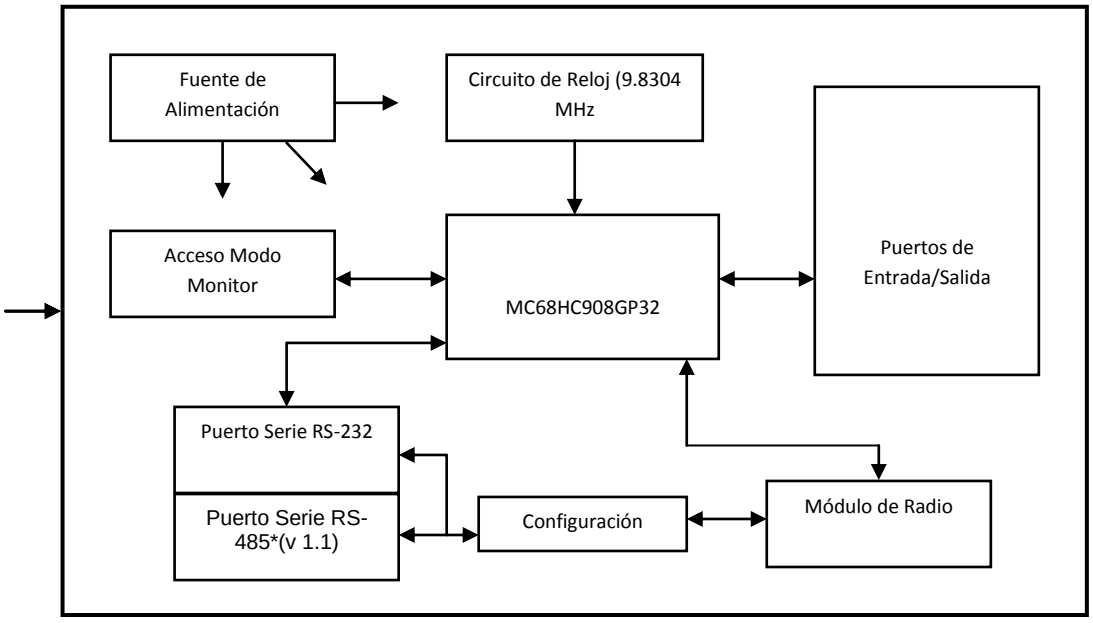


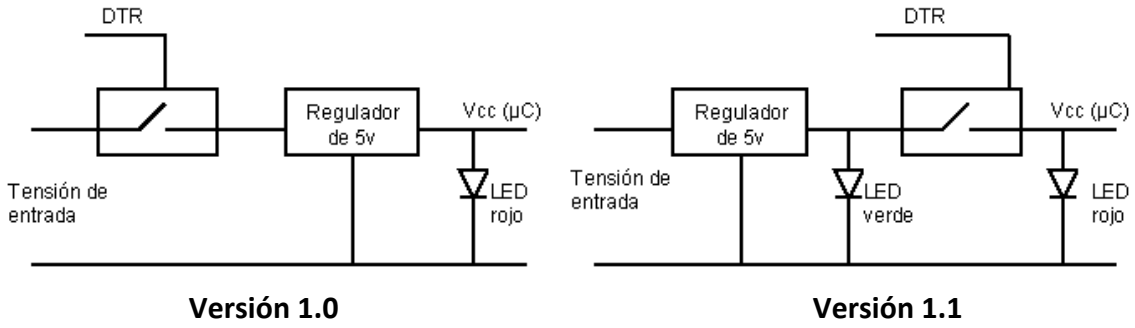
Figura AI.1. Diagrama de Bloques. GP_Bot

Como se puede observar en el diagrama de bloques anterior, el sistema de desarrollo contiene lo mínimo indispensable para que el micro-controlador pueda funcionar, pues se trata de un diseño de propósito general. Básicamente se trata del micro-controlador con todos sus puertos al exterior y sistemas de comunicación para su conexión con otros equipos.

Fuente de Alimentación: El sistema dispone de dos conectores para la entrada de alimentación, con idéntica funcionalidad: uno es una clema (permite la conexión de cable mediante tornillos) y otro es tipo jack. La tensión de entrada deberá estar entre 7 voltios y 13.5 voltios, para las placas de la versión 1.0, pues la fuente dispone de un

regulador interno de 5v que lo necesita. Para la versión 1.1 esta tensión de entrada puede ser a partir de 5,5 voltios y se recomienda que como máximo sea de 15 voltios. Esta tensión puede venir de una batería o bien de un transformador conectado a la red. La corriente que necesitará suministrar será función de los dispositivos que se conecten, pero se aconseja un mínimo de 250 mA.

Para que el microcontrolador entre en modo monitor debe producirse un arranque en frío, es decir, la tensión de alimentación debe bajar de 0.1 voltios. La fuente de alimentación dispone de un mecanismo que permite ese arranque en frío a través de la señal DTR del puerto serie del PC, controlado por el software de desarrollo. Si dicho software no soporta esa función, o en el puerto serie no aparece la señal DTR (es el caso de algunos portátiles) el arranque en frío se puede conseguir desconectando la alimentación o bien pulsando el botón de *"Power Up"* cuando así lo solicite el sistema de desarrollo. En este caso, se debe desconectar esta señal del puerto serie, en la versión 1.1. Para ello bastará con quitar el puente **JP3**. En la versión 1.0 se debe eliminar la conexión del pin 20 (en el conector DB25 del puerto serie).



Versión 1.0 **Versión 1.1**

Figura AI.2. Control de la fuente de alimentación por la señal DTR

Según los esquemas de la figura anterior, en la versión 1.0 el diodo LED puede no estar encendido aunque la placa esté correctamente alimentada, basta con que la señal DTR no esté activa. Esto puede inducir a error, por lo que se ha corregido en la versión 1.1. En esta versión, cuando la alimentación está conectada siempre luce el led verde. Si además la señal DTR esta activa también lucirá el led rojo. Ambos leds se han integrado en un led bicolor.

La iluminación de los led verde y rojo está bien en una primera fase de depuración, pero es posible que en ciertas circunstancias interese reducir el consumo del sistema, para lo cual se puede desconectar la iluminación de los leds. Esto se consigue eliminando el puente **JP4** (versión 1.1).

Circuito de Reloj: Oscilador típico basado en un cristal de 9.8304 MHz, lo que permite al micro-controlador generar las señales internas necesarias para su correcto funcionamiento.

Acceso al Modo Monitor: Permite la conexión del Cable Monitor. Incluye el hardware necesario para configurar la tarjeta como Clase I, según las clasificaciones de acceso indicadas por los diseñadores del software de desarrollo[1].

Micro-controlador MC68HC908GP32: Descrito anteriormente. El encapsulado empleado en este diseño es QFP *(quad flat pack*) de 44 pines. La elección de este encapsulado viene motivada por dos factores, en primer lugar es el más pequeño disponible. Esto ha permitido que el tamaño final de la tarjeta tenga unas reducidas dimensiones (70 x 65 mm). En segundo lugar es el que más pines tiene, con lo que se aumenta el número de bits de entrada/salida. En otros encapsulados hay ciertos pines que no están disponibles (ver la hoja de datos del micro-controlador[2]).

Puertos de Entrada/Salida: Acceso a todos los puertos de entrada/salida del micro-controlador, así como a ciertas señales de control y a la tensión de alimentación, tanto regulada (5v.) como sin regular (tensión de entrada).

Puerto Serie RS-232: Conexión serie normalizada, basada en el popular *driver* MAX232. Las líneas que controlan la comunicación serie son PTE0 y PTE1. Si se utiliza este puerto se deben insertar puentes en los *jumpers* de configuración etiquetados como JP1 y JP2 para la versión 1.0 o bien colocar ambos en la posición 1-2 para la versión 1.1. En caso de no utilizarse no se conectarán, con lo que el driver no consume nada. Hay que tener precaución con estas líneas, pues en cualquier caso siguen presentes en el bus de expansión.

Puerto serie RS-485: sólo en la versión 1.1. En esta caso se cambia el driver anterior (RS-232) por otro que maneja las señales eléctricas de este protocolo. Para utilizarlo hay que soldar el CI MAX3081 o equivalente en U6 y colocar los jumpers JP1 y JP2 en la posición 2-3. Se utilizará el conector J15 (que también se deberá soldar).

[1] Sistema de Desarrollo ICS, de Pemicro. Incluye ensamblador, programador, depurador y simulador. Se puede descargar en http://www.pemicro.com

[2] MC68HC908GP32, HCMOS Microcontroller Unit. Technical Data, Hoja de datos del microcontrolador disponible en la web de Motorola: http://www.mot-sps.com.

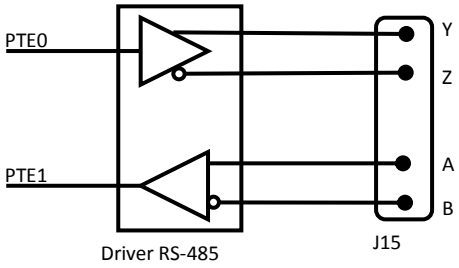
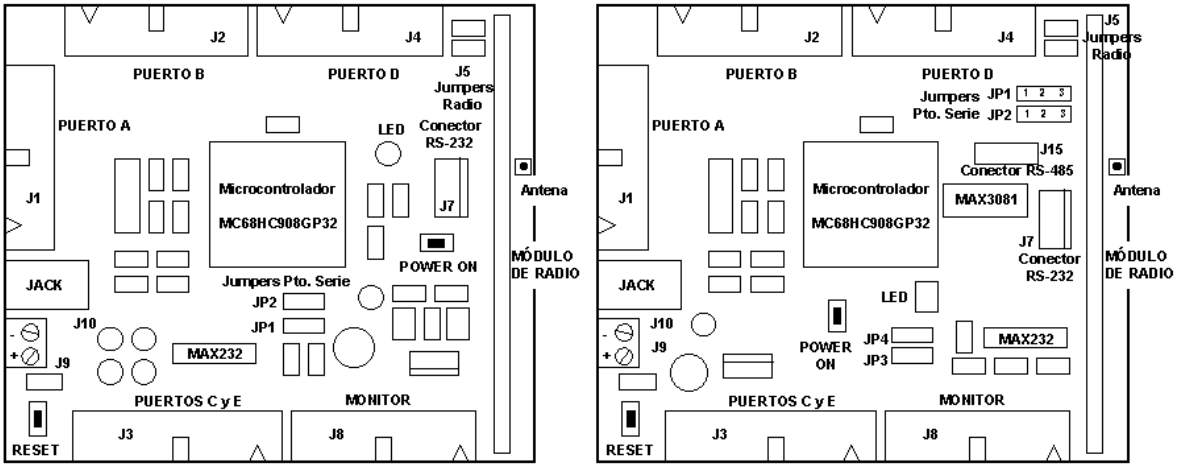


Figura AI.3. Diagrama de conexiones para el puerto RS-485

Módulo de Radio: Circuito híbrido para transmisión y recepción de datos digitales con antena única. Permite una comunicación *"half duplex"*. Utiliza el rango de frecuencia para radiomandos y radiocontrol de 433,92 MHz. La antena está formada por un hilo de cable conductor de 17 cm, aproximadamente, lo que proporciona un alcance de unos 200m sin obstáculos. Utiliza los pines PTD4 y PTD5 del microcontrolador. En caso de utilizar este módulo conviene desconectar estos pines del conector de expansión, lo que se consigue extrayendo los contactos ente los pines 1-2 y 3-4 del *jumper* J5, de forma que no haya posibles interferencias provenientes del exterior.

AI.2.3. Distribución de componentes en la tarjeta

A continuación se muestra la distribución de componentes en la tarjeta GP_Bot, resaltando los componentes más importantes, como son el microcontrolador, los conectores de expansión, los *jumpers* de configuración, el módulo de radio y los pulsadores de *Reset* y *Power On*.



Versión 1.0 **Versión 1.1**

Figura AI.4. Distribución de Componentes. GP_Bot

En la tarjeta hay dos pulsadores, etiquetados como RESET y POWER ON. La función de cada uno se detalla a continuación:

> **RESET**: Pone a nivel bajo la entrada de *Reset* del microcontrolador, produciéndose un *reset* externo. Se actualizan ciertos registros y el programa almacenado se ejecuta a partir de una dirección inicial conocida.
>
> **POWER ON**: Desconecta la alimentación del microcontrolador, con lo que al liberarlo entra en una secuencia de inicialización determinada ejecutando ciertos procesos que sitúan el microcontrolador en un estado inicial conocido, entre otros determina si debe entrar en modo monitor o no.

Para más detalle consultar el punto 19.4 *Reset and System Initialization* de la hoja de datos del microcontrolador.

En cuanto a la configuración de los jumpers, su función de describió en los apartados Puerto Serie RS-232, puerto serie RS-485 y Módulo de Radio del punto anterior.

El conector J7 es el utilizado para la comunicación serie RS-232. El cable necesario se describe en la figura AI.7. El conector J15 se usa para la comunicación serie a través del protocolo RS-485.

AI.2.3. Distribución de señales en los conectores externos.

La figura AI.5a muestra la forma en la que se distribuyen los pines en los conectores J1 a J4 y J8, que son tiras de pines paralelas en los que se insertará un conector para cinta plana de 10 o 16 hilos, según corresponda. Las figuras AI.5b y AI5c muestran los conectores J10 y J9 respectivamente, que se utilizan para la entrada de alimentación.

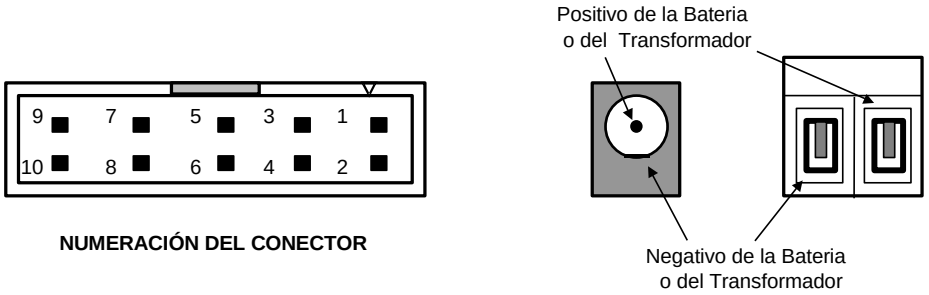


Figura AI.5. Interpretación de los conectores. GP_Bot

La distribución de señales en los pines de los conectores de expansión, vistos de frente, son los mostrados a continuación. El conector del puerto serie está visto desde arriba.

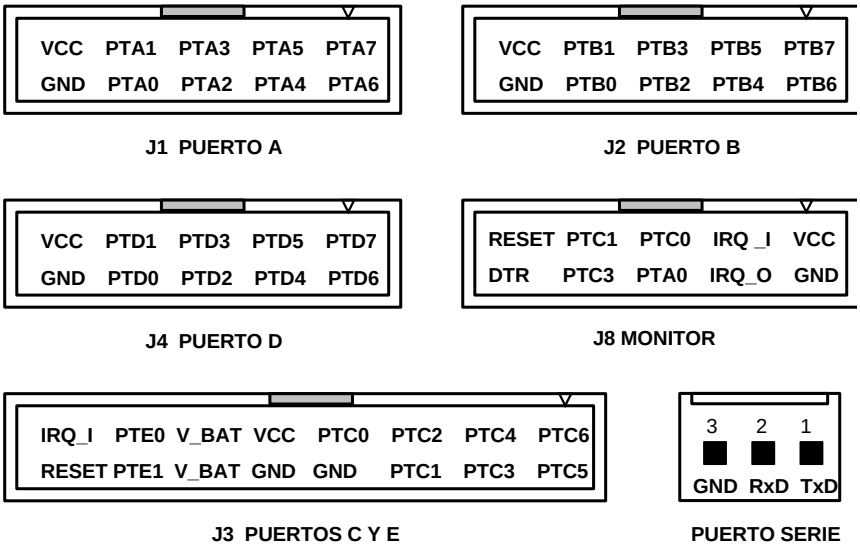


Figura AI.6. Conectores de Expansión. GP_Bot

NOTA: Los puertos A, C y D disponen internamente de una resistencia de pull-up programable bit a bit. Según las especificaciones de entrada del modo monitor, el pin PTA7 debe tener un nivel bajo por lo que se ha conectado una resistencia externa de pull-down de 10K. Por ello, este pin no debe tener nunca activada la resistencia de pull-up interna.
Igualmente, el pin PTA0 es utilizado por el programa monitor como entrada/salida de datos, por lo que se recomienda que se evite su uso, como mucho se podrá utilizar como salida, para que no interfiera en la comunicación con el monitor.

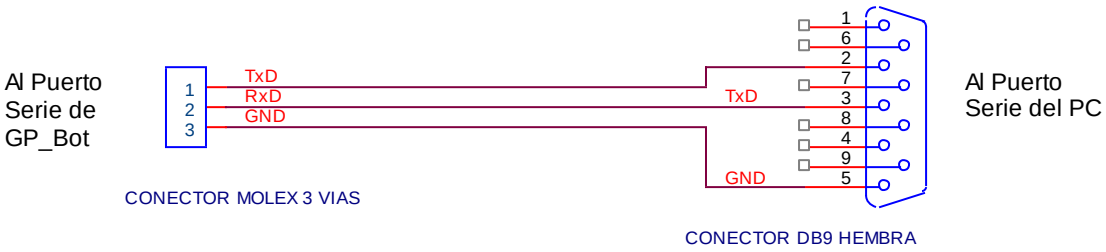


Figura AI.7. Cable para la conexión serie entre GP_Bot y un PC

(NOTA: este cable no es el utilizado para la programación y depurado)

AI.2.5. Conexión de la tarjeta al PC

El sistema de desarrollo está diseñado para que funcione de forma autónoma, pero para la programación y depuración del código es necesario que se conecte a un PC, entrando en el modo monitor del microcontrolador. Esta conexión se realiza a través del puerto serie del PC y es necesario una tarjeta de adaptación y conversión de señales, denominada GP_Mon, diseñada y desarrollada también en el transcurso de este proyecto.

El diagrama de conexión se muestra a continuación:

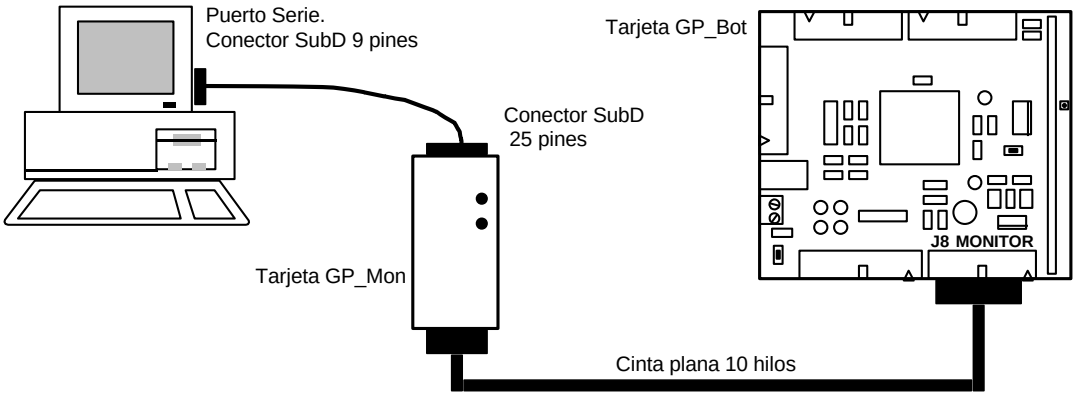


Figura AI.8. Diagrama de conexión PC - GP_Mon - GP_Bot

AI.3. Tarjeta GP_Mon y Cable Monitor

El Cable Monitor permite la conexión del PC con la tarjeta de desarrollo GP_Bot como un periférico de Clase I, con control del *Power On* a través de la línea DTR del puerto serie. Está compuesto de los siguientes elementos:

Cable Serie: Cable de 5 hilos que une el PC con el circuito de adaptación de niveles GP_Mon. En el lado del PC tiene un conector tipo SubD hembra de 9 pines y en el otro lado un SubD macho de 25 pines. El esquema de montaje se muestra en la figura AI.9.

Tarjeta GP_Mon: Contiene los circuitos necesarios para la adaptación de niveles del puerto serie así como para forzar la entrada del microcontrolador en modo monitor. Tiene dos microinterruptores nombrados como RTS y PTC3 cuyo uso es:

> **RTS**: Cuando está en ON, conecta la línea RTS del puerto serie con la entrada de *Reset* del microcontrolador. Algunos programas de desarrollo utilizan esta línea para enviar esta señal de inicialización al micro.
>
> **PTC3**: Determina la división de frecuencia del reloj externo (consultar la tabla 15.*1 Monitor Mode Signal Requirements and Options* de la hoja de datos del microcontrolador).
>
> **ON** (PTC3=0): Divide por 2
> **OFF** (PTC3=1): Divide por 4 (posición para la tarjeta GP_Bot)

La alimentación de la tarjeta se toma a través del cable que la une con la placa GP_Bot. Éste es un cable de cinta plana de 10 hilos.

Dispone de dos diodos LED, el rojo indica que está conectada la alimentación y el verde que se está produciendo una transferencia de datos serie.

ATENCIÓN: En las tarjetas Clase I, el software puede generar un *Power On*, es decir, desconecta la alimentación del microcontrolador y la vuelve a conectar. En el diseño realizado en la versión 1.0 se puede cortar la alimentación de toda la tarjeta, por lo que es posible que esté correctamente conectada la batería y no se encienda el LED, siempre que esté conectado el Cable Serie Clase I. Esto es debido a que en la señal DTR no está al nivel adecuado. Cuando se ejecute el software, éste lo pondrá en el nivel que corresponda.

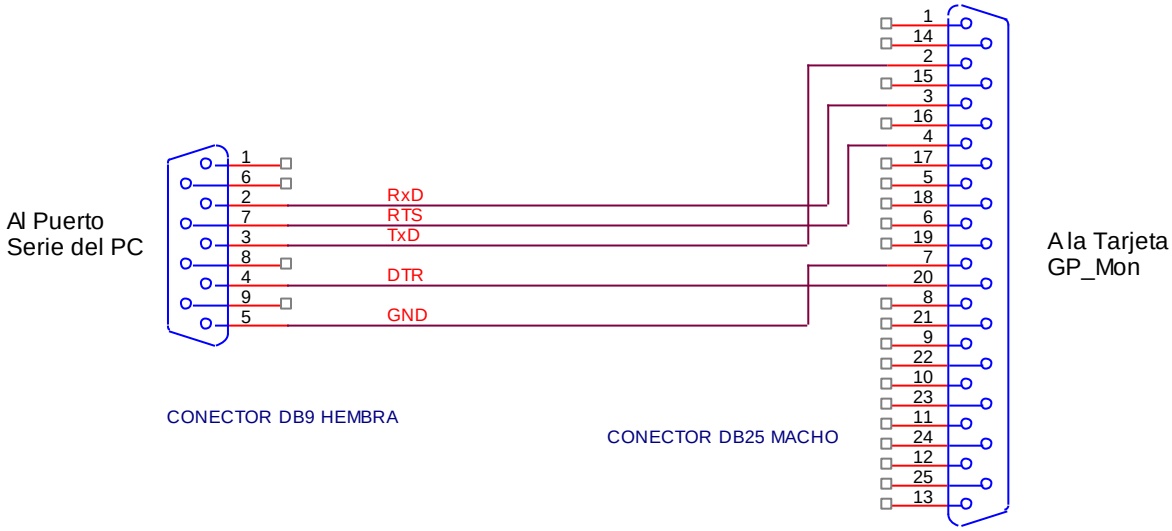


Figura AI.9. Cable Serie Clase I

AI.4. Tarjeta GP_Bot_Ifaz

AI.4.1. Características de la tarjeta

Esta tarjeta ha sido diseñada para actuar como interface entre la tarjeta GP_Bot y ciertos tipos de actuadores, como motores, sensores de infrarrojos y pulsadores. También dispone de una serie de pines de entrada/salida que pueden ser utilizados a voluntad del diseñador.

Las características principales son:

- Manejo de hasta 4 motores de corriente continua o dos motores paso a paso
- Conexión directa para 4 sensores de infrarrojos, incluyendo su polarización, con posibilidad de lectura tanto analógica como digital.
- 8 entradas analógicas que también pueden ser entrada/salidas digitales
- 4 señales de entrada/salida digital

- Permite un sistema de alimentación de los motores diferente al de la lógica (mayor potencia, inmunidad al ruido, etc).
- Fuente de alimentación regulada, con filtro LC

El tamaño de esta tarjeta es idéntico al de la GP_Bot, así como la colocación de los conectores que la unen a esta, lo que permite crear un conjunto bastante pequeño y compacto. La conexión se realiza uniendo los conectores J1, J2 y J3 de ambas tarjetas mediante cintas de cable plano y su conector correspondiente.

AI.4.2. Diagrama de bloques

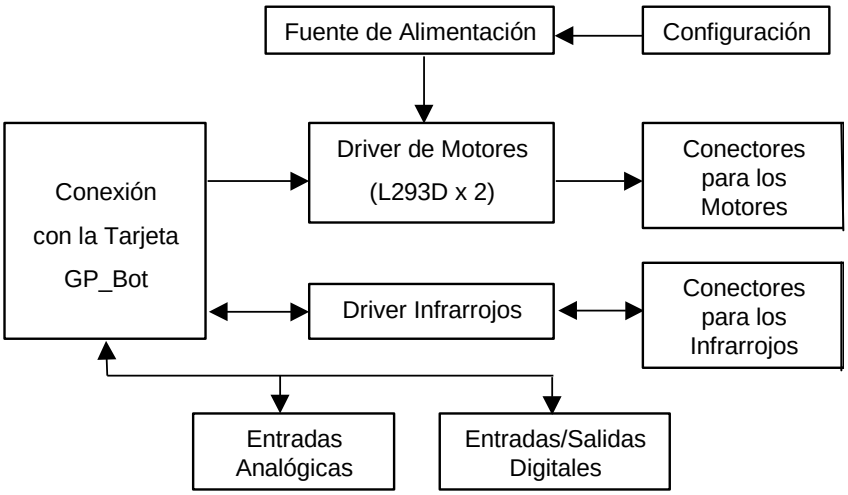


Figura AI.10. Diagrama de Bloques. GP_Bot_Ifaz

Fuente de Alimentación: Esta tarjeta dispone de una fuente de alimentación exclusivamente para alimentar a los *drivers* de motor, los integrados L293D. Está basada en el regulador lineal LM78xx, donde xx es la tensión de alimentación regulada que se quiere obtener. La Figura AI.11 muestra un diagrama de bloques con indicación de dos *jumpers* que permiten su configuración.

Jumper J6: Selecciona la tensión de entrada a la fuente

- Pines 1 y 2 conectados: seleccionada una fuente externa. Se conectará al conector J4.
- Pines 2 y 3 conectados: seleccionada la entrada de la batería conectada a la tarjeta GP_Bot. En este caso no se debe conectar nada en J4. Los motores se alimentará con la misma tensión que entregue dicha batería.

Jumper J7: Selecciona tensión regulada o no.

- Pines 1 y 2 conectados: seleccionada tensión regulada (hay que insertar el regulador necesario, de la familia 78xx).
- Pines 2 y 3 conectados: seleccionada tensión sin regular

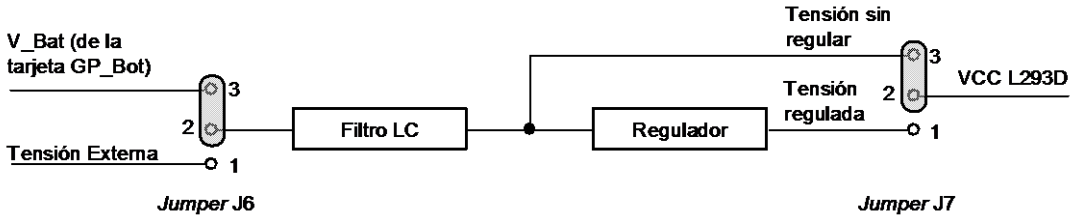


Figura AI.11. Diagrama de Bloques Fuente de Alimentación. GP_Bot_Ifaz

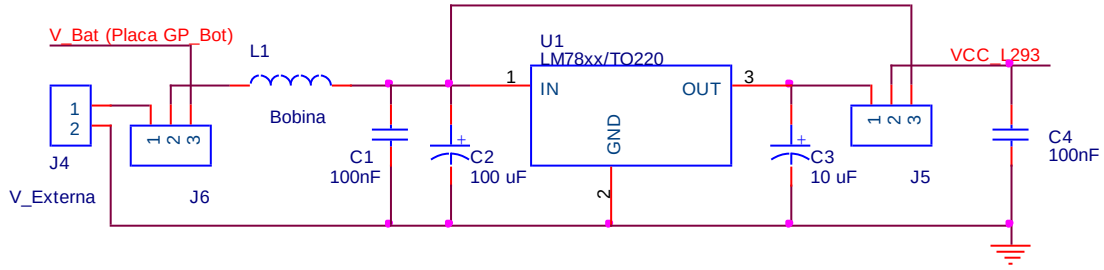


Figura AI.12. Esquema de la Fuente de Alimentación.

***Driver* de Motores**: Este bloque está formado por dos integrados del tipo L293D, con los diodos de protección integrados.

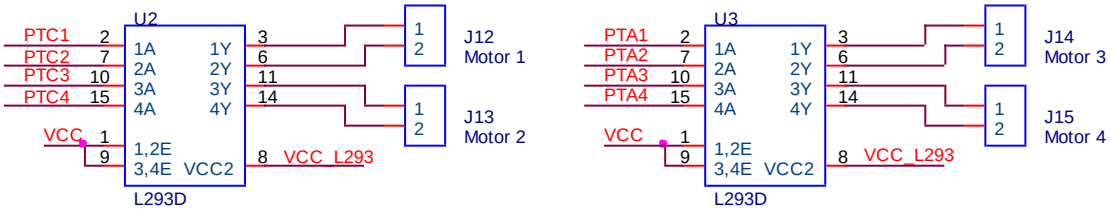


Figura AI.13. Conexiones de los *drivers* de los motores.

***Driver* Infrarrojos**: Diseñado para conectar los sensores CNY70 (o cualquier otro equivalente). Incluye las resistencias de excitación de los emisores de infrarrojos y las de *pull-up* (internas al microcontrolador). Cada sensor se puede leer desde un puerto digital o desde uno analógico.

Entradas analógicas: Conectados directamente al puerto B del microcontrolador permite la entrada de 8 canales analógicos. En caso de usar estas entradas se puede conectar el array de resistencias marcado como R1, con un valor de 10K.

Conexión con la tarjeta GP_Bot: Conectores de iguales características y posicionado que en la tarjeta GP_Bot, que permiten el intercambio de señales entre ambas. Conectan los puertos A, B, C y E y las tensiones de alimentación (Vcc=5v y la tensión de la batería).

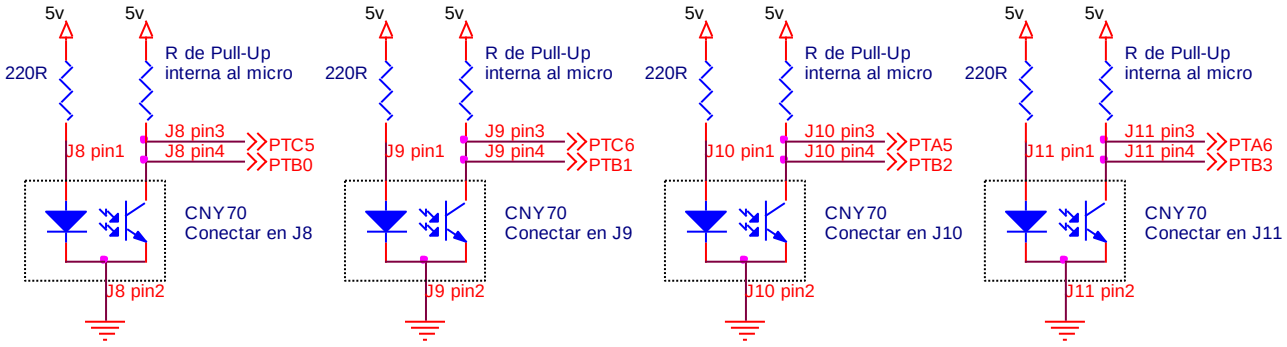


Figura AI.14. Conexiones de los sensores de infrarrojos.

AI.4.3. Distribución de componentes en la tarjeta

En la figuera AI.15 se muestra la distribución de los componentes de la tarjeta GP_Bot_Ifaz. Cabe destacar la función y la posición de los conectores:

- J1, J2 y J3: Conexión con la tarjeta GP_Bot
- J4: Entrada de tensión auxiliar para los motores
- J5 y J6: Jumpers de configuración de la fuente de alimentación, explicados anteriormente.
- J7: Puertos de entrada/salida del microcontrolador.
- J8 a J11: Conexión con los sensores de infrarrojos CNY70 (o cualquier otro equivalente)
- J12 a J15: Conexión de los motores

El lugar que ocupa el regulador opcional de la fuente está marcado como U1. Sólo se conectará si se desea una tensión regulada para los motores. El regulador será del tipo 78xx o equivalente.
El array de resistencias de *pull-up* del puerto B está etiquetado como R1. Sólo se conectará si se usa el puerto como entrada (analógica o digital).

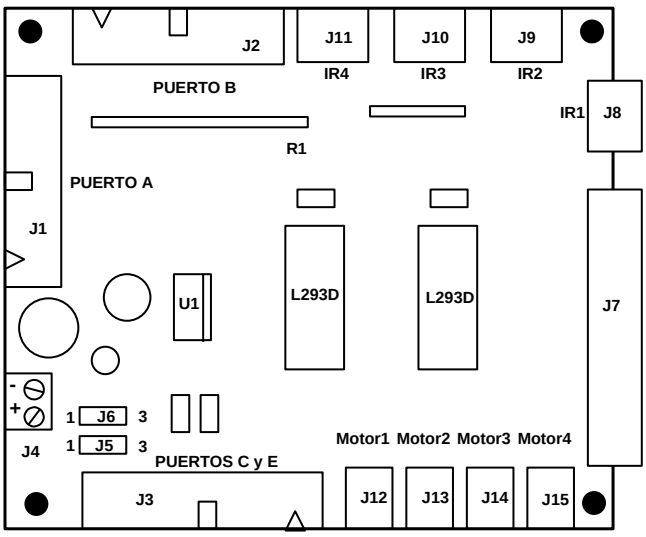


Figura AI.15. Distribución de Componentes. GP_Bot_Ifaz

AI.4.4. Distribución de señales en los conectores externos

La asignación de señales de cada conector es la siguiente:

- J1, J2 y J3. Son idénticos a los de la tarjeta GP_Bot, mostrados en la AI.6.
- J4: Visto de frente, el pin de la derecha es el positivo y el de la izquierda el negativo de la batería auxiliar.

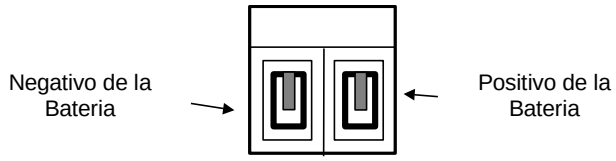


Figura AI.16. Conector de Alimentación. GP_Bot_Ifaz

- J7: Visto de frente (NOTA: en algunas placas este conector tiene la solapa de polarización en la parte inferior, la distribución de pines es la misma que se muestra aquí)

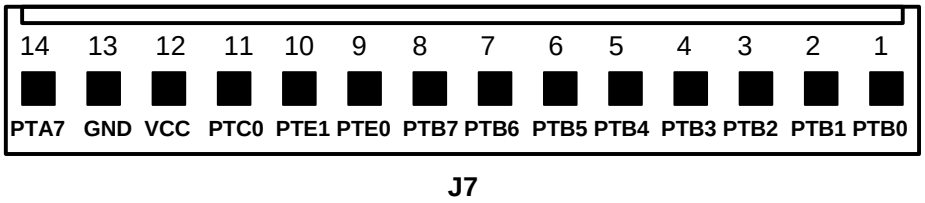


Figura AI.17. Puertos de Entrada/Salida. GP_Bot_Ifaz

- J8 a J11: Sensores de infrarrojos, vistos de frente (NOTA: en algunas placas este conector tiene la solapa de polarización en la parte inferior, la distribución de pines es la misma que se muestra aquí):

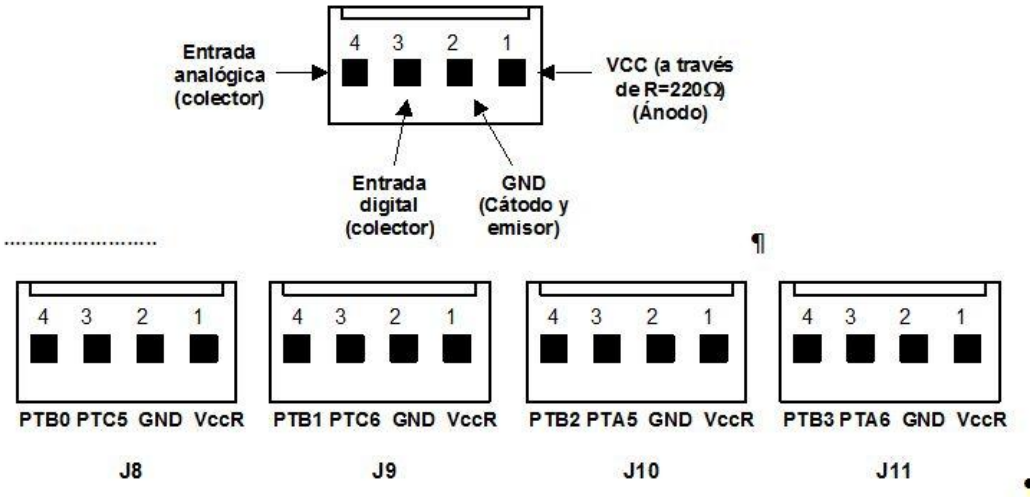


Figura AI.18. Conectores de los sensores de infrarrojos

- J12 a J15: Conectores de los motores vistos de frente, entre paréntesis se indica a qué pin está conectada cada salida. (NOTA: en algunas placas este conector tiene la solapa de polarización en la parte inferior, la distribución de pines es la misma que se muestra aquí)

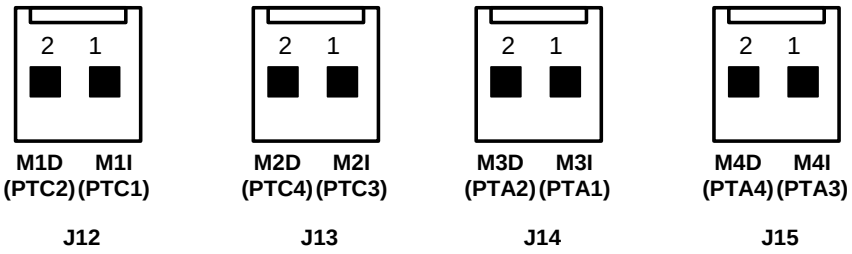


Figura AI.19. Conectores de los Motores

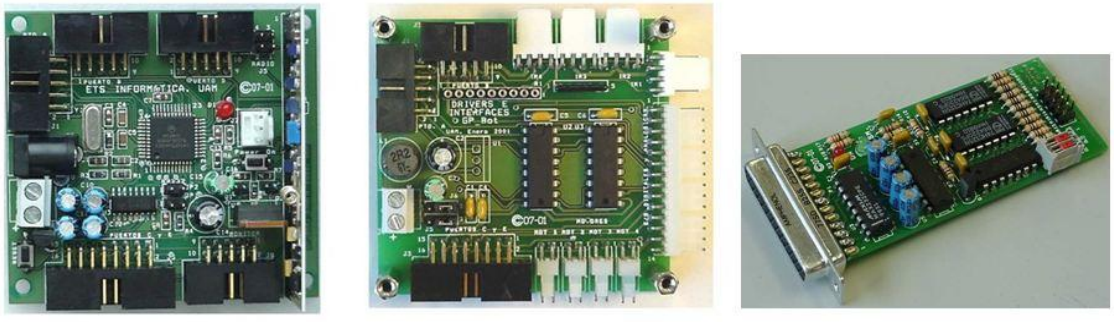
AI.4.5. Conexión con la tarjeta GP_Bot

La conexión mecánica entre las dos tarjetas se realizará mediante separadores atornillados en cada uno de los agujeros de cada esquina.

La conexión eléctrica se realiza a través de los conectores J1, J2 y J3 de ambas tarjetas. No es necesario conectar los tres, sólo los que se utilicen. Para ello se necesita un cable de cinta plana terminado con los conectores correspondientes. Prestar atención a la colocación de dicho conector, de forma que se unan los pines con idéntica numeración.

Recordar que la alimentación de los motores se puede tomar de la tarjeta GP_Bot o bien de una batería externa.

AI.5. Fotografías del sistema



(a) **(b)** **(c)**

Figura AI.20. GP_Bot (a), GP_Ifaz (b) y GP_Mon(c)

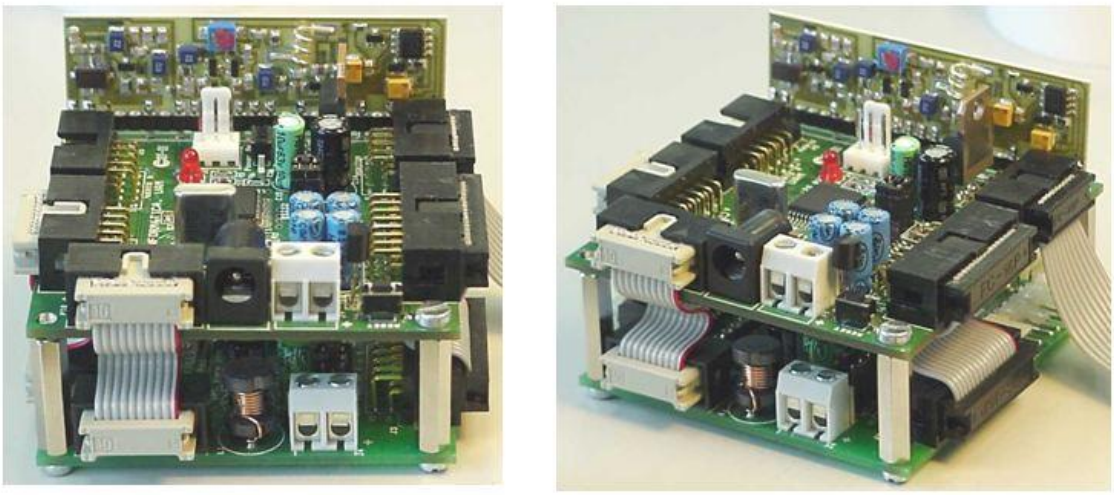


Figura AI.21. Conjunto GP_Bot completo, incluyendo el módulo de radio

Anexo II. Servidor Sensorial Transmisor-Receptor de radio RF-USB

A.II.

AII.1. Introducción

El módulo de RF-USB creado es capaz de mandar y recibir distintos comandos vía radio, a través del puerto USB. Sin alimentación extra necesaria, es capaz de crear una comunicación a través de radiofrecuencia entre dos estaciones de trabajo.

La comunicación del módulo con el puerto USB del procesador principal se realiza a través del chip FT232R, de la empresa FTDI. La comunicación a través de RF se realiza a través de los módulos de radio WM11, de la empresa DMD1.

El funcionamiento es sencillo: a través del puerto serie COM asociado al módulo, creado por el driver de FTDI, el procesador central envía y recibe los caracteres ASCII de los comandos implementados en el módulo de RF.

AII.2. Funcionamiento del Módulo RF-USB

AII.2.1. Instalación del Módulo

El fabricante FTDI proporciona los drivers necesarios para conectar el módulo con el procesador central en distintos sistemas operativos. Se pueden obtener versiones actualizadas, tanto para Linux como para Windows, en la página web:

http://www.ftdichip.com/Drivers/VCP.htm

A la hora de comenzar a utilizar el módulo para realizar comunicaciones FM, hay que definir en el host o sistema procesador central algunos parámetros iniciales de la comunicación serie a través del puerto COM. Primero hay que identificar el puerto COM que el FTDI ha configurado en el host, que figurará como “USB Serial Port (COM’X’)“, donde X es el identificador del puerto.

Una vez determinado el puerto COM del módulo RF-USB conectado, se pasa a configurar los parámetros de comunicaciones serie que se desea utilizar. La configuración de comunicación estándar de los módulos creados es: 57600,8,N,1

(57600 baudios, 8bits, sin paridad y un bit de parada). Una vez configurado el módulo y la comunicación, se podrá comenzar a utilizar los comandos preprogramados.

AII.2.2. Comandos a utilizar de los módulos DMD WM11.

Tal como se definió en el capítulo 3, el Servidor Sensorial contiene un gestor de comandos a través del cual se podrá acceder a todas las funcionalidades del mismo.

Los comandos para los módulos DMD-WM11 utilizados, se pueden encontrar en la hoja de datos del producto. Además, en su página web, podrá encontrarse más información actualizada. Los módulos utilizados se ejecutan con la versión 5.14 del firmware.

Mediante la realización de distintas pruebas se han detectado ciertas diferencias entre el resultado de unos comandos respecto de lo que figura en la hoja de datos del transmisor-receptor. Las diferencias más importantes encontradas entre el funcionamiento según el manual y el funcionamiento real, se detallan a continuación:

INI, IRF, RST: con cualquiera de estos comandos, el módulo sobre el que se aplique mandará un "sms broadcast" presentándose. De esta forma, cuando un módulo se conecte, reinicialice o resetee, los demás módulos recibirán un SMS quedando informados.

RFC: el canal por defecto seleccionado en esta versión de los módulos es el canal 12 (869.200 Mhz).

PWR: la potencia de emisión seleccionada por defecto en estas versiones es la 10 (+10dBm = 10mW).

SMS: puesto que el comando CMS está puesto a 16 (ver comando CMS), el número que aparece entre la dirección del módulo que envía el mensaje y el propio mensaje, es el valor de potencia en dBm del mensaje recibido.

SMI: Las pruebas de este comando y del comando SMQ, se han llevado a cabo con tres módulos: 137.005, 137.021 y 137.023.

Según la documentación, “este comando provoca la contestación aleatoria y automática de los equipos de la red con la RSSI recibida.“

Al poner a prueba esta instrucción, se obtiene que de los tres equipos, sólo uno envía la contestación al emisor. El tercer equipo no se la envía al emisor sino que se la envía al segundo. A continuación se muestran los datos recibidos por los distintos módulos:

Emisor: 137.005 Wlink11s 2A V:5.14, HCTLab RF-1

***Envía**: smi 1500*

Respuesta: < SMS 137.021 -068 -066,Wlink11s 2A V:5.14, HCTLab

Módulo: 137.023 Wlink11s 2A V:5.14,

< SMI 137.5, 1500: 1352

> SMI ID

Módulo: 137.021 Wlink11s 2A V:5.14, HCTLab RF-2

< SMI 137.5, 1500: 300

> SMI ID

< SMS 137.023 -062 -060,Wlink11s 2A V:5.14,

Tras varias pruebas con este comando, parece ser que siempre rotan en la respuesta, es decir, al emisor principal, en lugar de llegarle las dos respuestas, le llega la del módulo que antes la envía (en el ejemplo el módulo 137.021). Por otro lado, el último módulo manda su respuesta al que primero ha contestado, en lugar de al módulo emisor del comando (en este caso de ejemplo, el módulo 137.023 manda su respuesta al 132.021 en lugar de al módulo que la solicitó, el 137.005.

El funcionamiento con cuatro módulos no ha sido probado.

SMQ: Para este comando, el funcionamiento entre los tres módulos visto desde el módulo que envía la pregunta debería ser el siguiente:

Módulo 137.005 envía: SMQ 000.000 "¿dónde estáis?"

Respuestas:

<SMS 137.021 -0.52 -0.54 → "el módulo 137.021 responde enviando la potencia que ha recibido"

<SMS 137.023 -0.65 -0.64 → "el módulo 137.023 responde enviando la potencia que ha recibido"

Sin embargo, al hacer la prueba con tres módulos el resultado es el siguiente caso que se adjunta mediante las capturas de pantalla de la información recibida:

137.005 Wlink11s 2A V:5.14, HCTLab RF-1

Envia: smq 000.000 ¿Donde estais?

Recepción: < SMS 137.023 -041 -037, Ok

137.023 Wlink11s 2A V:5.14,

< SMQ 137.005 -037 ¿Donde estais?

137.021 Wlink11s 2A V:5.14, HCTLab RF-2

< SMQ 137.005 -064 ¿Donde estais?

En conclusión, ambos módulos reciben el sms, pero sólo uno de ellos contesta (se repite siempre, sólo contesta uno, NUNCA los otros dos), enviando la potencia recibida al emisor.

AII.3. Esquema electrónico del Servidor Sensorial

El esquema del Servidor Sensorial se muestra en la figura AII.1. Básicamente está compuesto por un conversor USB a serie y el módulo de radio.

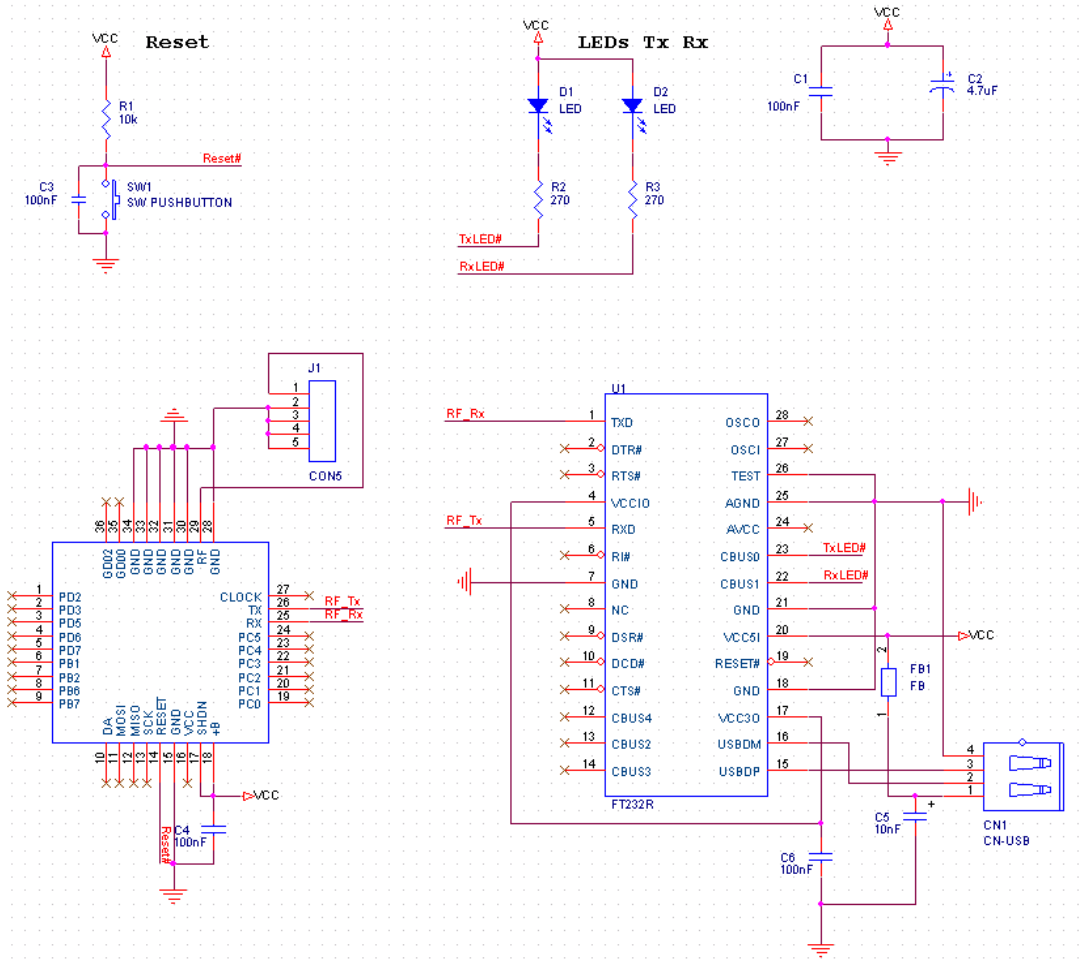


Figura AII.1. Esquema electrónico del SS. RF-USB

AII.4. Módulo de radio WM11

El módulo de radio, es el modelo WM11, de la empresa DMD, y se muestra en la figura AII.2. WM11 es una familia de módulos LPR (Low Power RF) con modulación GFSK y FSK, bi-direccionales y unidireccionales, disponibles en varias bandas según la aplicación. Una de las más utilizadas para este módulo son las bandas ICM (Industrial Científica y Médica) Europea (EN 300 220) de 434Mhz, 868Mhz y 902-927Mhz (FCC47) para EEUU. También hay modelos disponibles en la banda ICM 2400Mhz, muy aceptada en todo el mundo. Contiene un avanzado microcontrolador con 16kb-32Kb de memoria flash de programa (según modelos), con el software necesario para telemando y transmisión de datos sin necesidad de conocimientos avanzados de radio.

Las principales características de este módulo son:

- Frecuencia 868.100 869.900 MHz, 10mW.

- Modulación RF: 2FSK, 38kbaudios. Semiduplex.
- 79 Canales RF, según configuración.
- De acuerdo con EN 300 220.
- Precisión: PLL controlado por cristal +-5ppm.
- Sensibilidad RF: según modulación de –99dBm a -107dBm.
- Potencia RF salida programable desde –50dBm a +10dBm. WM11_8XRM
- Microcontrolador con 16k flash programa, incluido.
- Conversor D/A 10b opcional.
- Disponible ci adaptador compatible con la serie W868ATxx.
- Protocolo en red Unibus11W. Direccionamiento IP 16 bits.
- Puertos Entrada / Salida digitales y analógicos.
- Puerto serie asíncrono 2k4 - 250Kb
- Uso sencillo, tiempo muy corto de desarrollo.
- Alimentación: regulador integrado. 3V3 a 5.5V
- Consumos estándar: @RX <1 a 16mA, @TX 35mA
- Consumo @RX módulos ultra bajo consumo: <10uA
- Alcance: interiores de 100m a 200m.
- Tamaño 25.4 x 25.4 x 2.5mm
- Alcance: exteriores hasta 20Km según tipo de antena y cond.
- Dimensiones: 25.4 x 25.4 x 2.5mm.
- Muy buena relación prestaciones / Precio.
- Comandos incluidos:
    - SMS. Mensajes cortos texto.
    - SMB. Mensajes datos binarios.
    - SMI. Identificación en red

- NAM. Nombre amigable
- Configuración y test.
- Selección de canal RF.
- Control potencia RF y sensibilidad.
- Transponder activo
- Repetidor simple.

AII.4.1. Comandos del módulo WM11

Control. El módulo WM11 se controla y configura mediante secuencias simples de comandos, vía la interfase serie de forma muy similar al control de un módulo GSM de telefonía móvil. Hay comandos que son opcionales según la versión de software del módulo de RF.

Entrada de comandos. Los comandos se pueden introducir manualmente desde un PC usando el hiperterminal de Windows configurado por ejemplo a 57600b, 8, N, 1 ó con programas de aplicación tanto desde un microcontrolador como desde un PC. Se debe tener en cuenta que la velocidad en baudios es programable. Si se usan micro-controladores ó programas en PC, hay que desactivar la opción de ECO (comando "ECO0"), para lograr unas comunicaciones más fluidas en el programa de aplicación, ya que el eco es útil cuando se usa un terminal en pruebas y en cambio dificulta el desarrollo de las comunicaciones en la aplicación real final.

Los comandos se pueden ejecutar de forma remota en cualquier WM11, enviado un mensaje al sistema (Ver comando ATSMS), con el contenido del comando local que se quiere ejecutar. Esto proporciona una gran flexibilidad al sistema ya que puede cambiar parámetros como la potencia de emisión de forma remota a cualquier modulo de la red. También sirve para que un modulo funcione como repetidor.

Multitarea. El modulo de RF permite atender el sistema de radio al mismo tiempo que entran los comandos, es decir, que un equipo puede ejecutar un comando de lectura de un canal A/D que otro equipo remoto le solicitó, mientras le entra un mensaje SMS

por el puerto serie para un tercer equipo. Algunos tipos de comandos pueden ser incompatibles con este modo de funcionamiento.

Test de comunicaciones. WM11 dispone de un comando para asegurar la calidad de *slared* ó instalación final, realizando un test (ver comando SMQ). Se podrán verificar así los niveles óptimos de potencia de RF y los niveles óptimos de detección de portadora CD, tanto de su equipo como del remoto. Este test sirve también para verificar la calidad relativa de las antenas y cables de antena de los equipos en la instalación final, sobre todo si se usan antenas direccionales de alta ganancia. En equipos portátiles este test es de gran ayuda pues puede verificar la calidad del enlace en cualquier posición de la celda. En equipos en los que quiere alcanzar la distancia máxima este test dará a conocer hasta dónde puede llegar con sus equipos y configuración actual de antenas

AII.4.1.1. Sintaxis de los comandos V:3.0

- Un comando se compone de 3 caracteres, seguido por un espacio ó “?” ó “=”, dirección IP, separador, datos, CR
- Una línea de comandos puede terminar con <CR>, <LF>, <NULL> ó tiempo en caso de SMB con datos binarios.
- No se pueden combinar varios comandos en la misma línea. El buffer de comandos es de 80 bytes.
- Un SMS no puede contener más de 57 bytes de datos.
- Si la dirección IP es la propia, el comando es para sí mismo.
- La dirección IP y los datos son opcionales según el tipo de comando.

Ejemplo de comando típico:

SMS 104.245,Mensaje de pruebas <CR>

Los paquetes de datos en RF que provoca el transmisor, no tienen siempre respuesta por parte del receptor, dejando a la aplicación final la responsabilidad de la respuesta y verificación del enlace. Esto es así porque multiplica por dos la interactividad y velocidad efectiva de los módulos sobre todo con mensajes SMS.

Para saber que el equipo receptor ha recibido el mensaje, la aplicación final, si lo necesita, debe de disponer de un pequeño protocolo de respuestas y repeticiones parecido al típico protocolo Zmodem, para transmisión de ficheros.

Figura AII.2. Módulo WM11

El usuario tan sólo necesita conocer los comandos de la familia de Módulos WM11. Para usar el WM11, basta conectar la antena, alimentación sin regular de 3,3V a 6V y conectar TX y RX al microcontrolador ó mediante un convertidor de nivel al puerto COM de un PC.

Tabla AII.1. Listado de comandos del módulo WM11 implementados en esta versión.

Nemónico	Descripción	Valor Inicial
INF	Muestra la versión IP del producto	051.005 Wlink11s 2A V:3.8
INI	Inicializa el sistema	--
IRF	Inicializa subsistemas de radio y test RF	--
RST	Reset sistema	--
	Configuración General	
NAM	Nombre del equipo	--
ECO	Eco puerto serie. Grava siempre en EEPROM	X
BAU	Baudios puerto serie. Grava siempre en EEPROM	57.600
CPS	Configuración puerto serie	N,000
	Red	

LEA	Lista de equipos autorizados	No activada
LNK	Enlace (link) con equipo	No enlace
	Configuración RF	
RFC	Canal RF	1
FEC	Corrección errores en RF	3
PWR	Potencia transmisión RF	+7 dBm
NCD	Nivel detección portadora RF (Rssi)	Desactivado, 255
REP	Repetidor simple activación/desactivación	Activado
RSI	Devuelve valor portadora RF (Rssi)	-10 a 117 dBm
	Ports Entrada - Salida	
OUT	Escribe en puerto salida (8 bits)	0
INP	Lee en puerto entrada/salida (8 bits)	0
CFG	Configura puerto (8 bits) entradas-salidas	--
	SMS de comunicaciones RF	
SMS	Envía mensaje de texto/ binario. Max 54 bytes	
SMC	Mensaje al sistema de comandos remoto	
SMQ	Control calidad del mensaje	
SMI	Mensaje de petición de identificación equipos en red	
CMS	Configuración recepción SMS	
TRZ	Activa traza RF test recepción calidad SMS	Desactivado
	Varios	
E2P	Lee o escribe en memoria EEPROM	

La figura AII.3 muestra una imagen de las dos implementaciones realizadas del Servidor Sensorial, donde destacan las antenas instaladas.

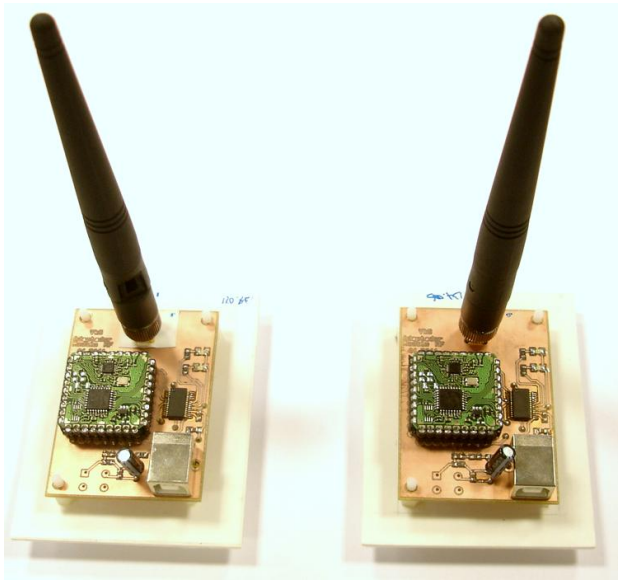


Figura AII.3. Servidor Sensorial RF_USB.

La figura AII.4 muestra la tarjeta fabricada y montada, con más detalle.

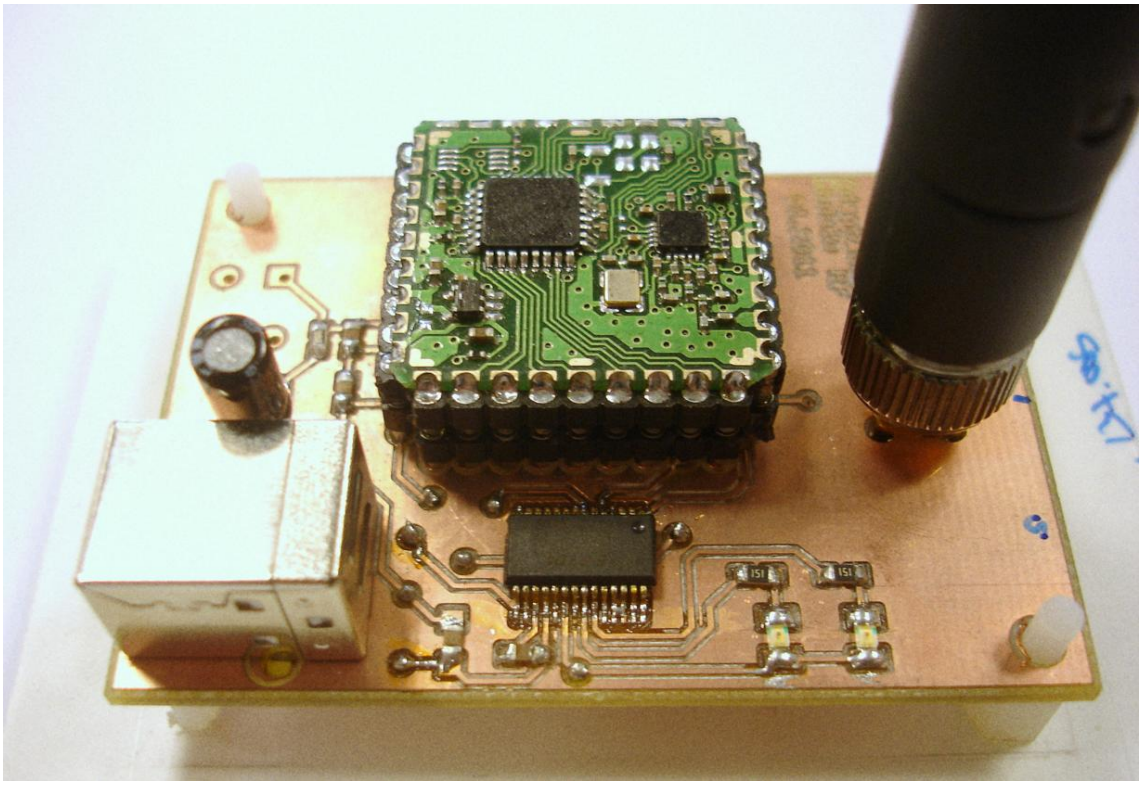


Figura AII.4. PCB fabricado y montado del S.S. RF-USB

Anexo III. Información para el manejo del driver de motor paso a paso de Pilot

A.III.

Para la implementación de la plataforma descrita en el punto 4.2 del capítulo 4 se utilizó el driver hardware de motor *Pilot Motion Processor* MC3410, de PMD [4.9]. Es un procesador que ofrece funciones de generación de trayectorias y control relacionado con los motores paso a paso. La comunicación del chip con el procesador central se realiza a través de una comunicación serie. Se utilizan comandos cortos enviados y recibidos como secuencias de bits o palabras. Estos paquetes contienen el código de instrucción que indica el tipo de acción a realizar. Pueden contener datos, tanto en el envío como en la respuesta. Al no disponer de código para Linux, es decir, todas las librerías que ofrecía el fabricante sólo se podían utilizar desde el sistema operativo Windows, como parte de este trabajo de tesis se escribió una librería completa para el acceso desde Linux. A continuación se incluye la lista de comandos implementados y un fichero de cabecera de c (librobot.h) a modo de ejemplo del código escrito.

AIII.1. Lista de comandos

Lista de Comandos

```
NoOperation
SetMotorLimit
GetMotorLimit
SetMotorBias
SetPosition
SetVelocity
SetJerk
SetGearRatio
Update
GetCommandedPosition
GetCommandedVelocity
SetKp
SetKi
SetKd
SetKvff
GetPhaseAngle
GetMotorBias
SetInterruptMask
GetEventStatus
ResetEventStatus
GetCaptureValue
GetActualPosition
SetSampleTime
Reset
GetCurrentMotorCommand
GetTime
ClearPositionError
GetPosition
GetVelocity
GetAcceleration
SetActualPosition
GetKp
GetKi
GetKd
GetKvff
GetInterruptMask
GetJerk
GetGearRatio
GetSampleTime
GetMotorCommand
SetStartVelocity
GetStartVelocity
GetOutputMode
SetPhaseInitializeTime
SetPhaseCounts
SetPhaseOffset
```

```
SetMotorCommand
InitializePhase
GetPhaseOffset
GetPhaseInitializeTime
GetPhaseCounts
SetLimitSwitchMode
GetLimitSwitchMode
WriteIO
ReadIO
SetPhaseAngle
SetNumberPhases
GetNumberPhases
SetAxisMode
GetAxisMode
SetDiagnosticPortMode
GetDiagnosticPortMode
SetSerialPortMode
GetSerialPortMode
SetEncoderModulus
GetEncoderModulus
GetVersion
SetAcceleration
SetDeceleration
GetDeceleration
SetKaff
GetKaff
SetIntegrationLimit
GetIntegrationLimit
SetPositionErrorLimit
GetPositionErrorLimit
GetPositionError
GetIntegral
GetDerivative
SetDerivativeTime
GetDerivativeTime
SetKout
GetKout
SetProfileMode
GetProfileMode
SetSignalSense
GetSignalSense
GetSignalStatus
GetHostIOError
GetActivityStatus
GetCommandedAcceleration
SetTrackingWindow
GetTrackingWindow
SetSettleTime
GetSettleTime
ClearInterrup
GetActualVelocity
SetGearMaster
GetGearMaster
SetTraceMode
GetTraceMode
SetTraceStart
GetTraceStart
SetTraceStop
GetTraceStop
SetTraceVariable
GetTraceVariable
SetTracePeriod
GetTracePeriod
GetTraceStatus
GetTraceCount
SetSettleWindow
GetSettleWindow
SetActualPositionUnits
GetActualPositionUnits
SetBufferStart
GetBufferStart
SetBufferLength
GetBufferLength
SetBufferWriteIndex
GetBufferWriteIndex
SetBufferReadIndex
GetBufferReadIndex
WriteBuffer
ReadBuffer
SetBufferFunction
GetBufferFunction
SetStopMode
GetStopMode
SetAutoStopMode
GetAutoStopMode
SetBreakpoint
GetBreakpoint
SetBreakpointValue
GetBreakpointValue
SetCaptureSource
GetCaptureSource
SetEncoderSource
GetEncoderSource
SetMotorMode
GetMotorMode
SetEncoderToStepRatio
GetEncoderToStepRatio
SetOutputMode
GetInterruptAxis
SetCommutationMode
GetCommutationMode
SetPhaseInitializeMode
GetPhaseInitializeMode
SetPhasePrescale
GetPhasePrescale
SetPhaseCorrectionMode
GetPhaseCorrectionMode
GetPhaseCommand
SetMotionCompleteMode
GetMotionCompleteMode
SetAxisOutSource
GetAxisOutSource
ReadAnalog
SetSynchronizationMode
GetSynchronizationMode
AdjustActualPosition
GetChecksum
```

AIII.2. Fichero de cabeceras de definiciones (*librobot.h*).

```
#ifndef _LIBER1_H
#define _LIBER1_H

#include "stddef.h"
#include "libserie.h"
#define IZDA 0x00
#define DRCHA 0x01

#define TRAPEZOIDAL 0x00
#define VELOCITY_CONTOURING 0x01
#define S_CURVE 0x02
#define EXTERNAL 0x04
#define ON 0x01
#define OFF 0x00

//Comandos
#define NoOperation 0x00
#define SetMotorLimit 0x06
#define GetMotorLimit 0x07
#define SetMotorBias 0x0F
#define SetPosition 0x10
#define SetVelocity 0x11
#define SetJerk 0x13
#define SetGearRatio 0x14
#define Update 0x1A
#define GetCommandedPosition 0x1D
#define GetCommandedVelocity 0x1E
#define SetKp 0x25
#define SetKi 0x26
#define SetKd 0x27
#define SetKvff 0x2B
#define GetPhaseAngle 0x2C
#define GetMotorBias 0x2D
#define SetInterruptMask 0x2F
#define GetEventStatus 0x31
#define ResetEventStatus 0x34
#define GetCaptureValue 0x36
#define GetActualPosition 0x37
#define SetSampleTime 0x38
#define Reset 0x39
#define GetCurrentMotorCommand 0x3A
#define GetTime 0x3E
#define ClearPositionError 0x47
#define GetPosition 0x4A
#define GetVelocity 0x4B
#define GetAcceleration 0x4C
#define SetActualPosition 0x4D
#define GetKp 0x50
#define GetKi 0x51
#define GetKd 0x52
#define GetKvff 0x54
#define GetInterruptMask 0x56
#define GetJerk 0x58
#define GetGearRatio 0x59
#define GetSampleTime 0x61
#define GetMotorCommand 0x69
#define SetStartVelocity 0x6A
#define GetStartVelocity 0x6B
#define GetOutputMode 0x6E
```

```
#define SetPhaseInitializeTime 0x72
#define SetPhaseCounts 0x75
#define SetPhaseOffset 0x76
#define SetMotorCommand 0x77
#define InitializePhase 0x7A
#define GetPhaseOffset 0x7B
#define GetPhaseInitializeTime 0x7C
#define GetPhaseCounts 0x7D
#define SetLimitSwitchMode 0x80
#define GetLimitSwitchMode 0x81
#define WriteIO 0x82
#define ReadIO 0x83
#define SetPhaseAngle 0x84
#define SetNumberPhases 0x85
#define GetNumberPhases 0x86
#define SetAxisMode 0x87
#define GetAxisMode 0x88
#define SetDiagnosticPortMode 0x89
#define GetDiagnosticPortMode 0x8A
#define SetSerialPortMode 0x8B
#define GetSerialPortMode 0x8C
#define SetEncoderModulus 0x8D
#define GetEncoderModulus 0x8E
#define GetVersion 0x8F
#define SetAcceleration 0x90
#define SetDeceleration 0x91
#define GetDeceleration 0x92
#define SetKaff 0x93
#define GetKaff 0x94
#define SetIntegrationLimit 0x95
#define GetIntegrationLimit 0x96
#define SetPositionErrorLimit 0x97
#define GetPositionErrorLimit 0x98
#define GetPositionError 0x99
#define GetIntegral 0x9A
#define GetDerivative 0x9B
#define SetDerivativeTime 0x9C
#define GetDerivativeTime 0x9D
#define SetKout 0x9E
#define GetKout 0x9F
#define SetProfileMode 0xA0
#define GetProfileMode 0xA1
#define SetSignalSense 0xA2
#define GetSignalSense 0xA3
#define GetSignalStatus 0xA4
#define GetHostIOError 0xA5
#define GetActivityStatus 0xA6
#define GetCommandedAcceleration 0xA7
#define SetTrackingWindow 0xA8
#define GetTrackingWindow 0xA9
#define SetSettleTime 0xAA
#define GetSettleTime 0xAB
#define ClearInterrupt 0xAC
#define GetActualVelocity 0xAD
#define SetGearMaster 0xAE
#define GetGearMaster 0xAF
#define SetTraceMode 0xB0
#define GetTraceMode 0xB1
#define SetTraceStart 0xB2
#define GetTraceStart 0xB3
#define SetTraceStop 0xB4
```

```
#define GetTraceStop 0xB5
#define SetTraceVariable 0xB6
#define GetTraceVariable 0xB7
#define SetTracePeriod 0xB8
#define GetTracePeriod 0xB9
#define GetTraceStatus 0xBA
#define GetTraceCount 0xBB
#define SetSettleWindow 0xBC
#define GetSettleWindow 0xBD
#define SetActualPositionUnits 0xBE
#define GetActualPositionUnits 0xBF
#define SetBufferStart 0xC0
#define GetBufferStart 0xC1
#define SetBufferLength 0xC2
#define GetBufferLength 0xC3
#define SetBufferWriteIndex 0xC4
#define GetBufferWriteIndex 0xC5
#define SetBufferReadIndex 0xC6
#define GetBufferReadIndex 0xC7
#define WriteBuffer 0xC8
#define ReadBuffer 0xC9
#define SetBufferFunction 0xCA
#define GetBufferFunction 0xCB
#define SetStopMode 0xD0
#define GetStopMode 0xD1
#define SetAutoStopMode 0xD2
#define GetAutoStopMode 0xD3
#define SetBreakpoint 0xD4
#define GetBreakpoint 0xD5
#define SetBreakpointValue 0xD6
#define GetBreakpointValue 0xD7
#define SetCaptureSource 0xD8
#define GetCaptureSource 0xD9
#define SetEncoderSource 0xDA
#define GetEncoderSource 0xDB
#define SetMotorMode 0xDC
#define GetMotorMode 0xDD
#define SetEncoderToStepRatio 0xDE
#define GetEncoderToStepRatio 0xDF
#define SetOutputMode 0xE0
#define GetInterruptAxis 0xE1
#define SetCommutationMode 0xE2
#define GetCommutationMode 0xE3
#define SetPhaseInitializeMode 0xE4
#define GetPhaseInitializeMode 0xE5
#define SetPhasePrescale 0xE6
#define GetPhasePrescale 0xE7
#define SetPhaseCorrectionMode 0xE8
#define GetPhaseCorrectionMode 0xE9
#define GetPhaseCommand 0xEA
#define SetMotionCompleteMode 0xEB
#define GetMotionCompleteMode 0xEC
#define SetAxisOutSource 0xED
#define GetAxisOutSource 0xEE
#define ReadAnalog 0xEF
#define SetSynchronizationMode 0xF2
#define GetSynchronizationMode 0xF3
#define AdjustActualPosition 0xF5
#define GetChecksum 0xF8
```

```
typedef struct{
      ls_puerto *puerto;
}lr_robot;

lr_robot *lr_inicializa(char *puerto);
void lr_finaliza(lr_robot *robot);
int lr_envia_recibe(lr_robot *robot,BYTE * comando, int
longitud_comando,BYTE * respuesta, int longitud_respuesta);
int lr_envia_recibe_commando(lr_robot *robot,BYTE * comando, int
longitud_comando,BYTE * respuesta, int longitud_respuesta);
int lr_crea_paquete(BYTE direccion, BYTE *mensaje, int longitud,BYTE *
paquete);
int lr_recibe_paquete(lr_robot *robot,BYTE *paquete, int longitud);
int lr_checksum(BYTE *mensaje, int longitud, int write);
void lr_imprime_traza(BYTE *traza, int longitud);

int lr_cmd(lr_robot *robot,BYTE direccion,BYTE comando);
int lr_set32(lr_robot *robot,BYTE direccion,BYTE comando,UNSIGNED_32
value);
int lr_set16(lr_robot *robot,BYTE direccion,BYTE comando,UNSIGNED_16
value);
int lr_get32(lr_robot *robot,BYTE direccion,BYTE comando,UNSIGNED_32
*value);
int lr_get16(lr_robot *robot,BYTE direccion,BYTE comando,UNSIGNED_16
*value);

int lr_inipilot(lr_robot *robot,BYTE direccion);
int lr_mover_trapezoide(lr_robot *robot,BYTE direccion, SIGNED_32
posicion,
      SIGNED_32 velocidad, UNSIGNED_32 aceleracion, UNSIGNED_32
deceleracion);
int lr_mover_velocity(lr_robot *robot,BYTE direccion, SIGNED_32
velocidad,
      UNSIGNED_32 aceleracion, UNSIGNED_32 deceleracion);

#endif /* _LIBER1_H */
```

Anexo IV. Ficheros generados

A.IV.

A continuación se muestran los ficheros fuente generados para el desarrollo de todas las aplicaciones creadas para el Sistema GdRBot, servidores, clientes, pruebas, código para los servidores sensoriales, etc. Se ha generado utilizando el comando “dir” de MS-DOS sobre el proyecto raíz del trabajo de tesis.

```
Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src

05/08/2011  10:39    <DIR>          Driver_Pilot
05/08/2011  10:39    <DIR>          findcolor
05/08/2011  10:39    <DIR>          gdrbot
05/08/2011  10:39    <DIR>          gpbot
               0 archivos              0 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\Driver_Pilot

05/08/2011  10:39               896 command.c
05/08/2011  10:39             2.536 ejemplo.c
05/08/2011  10:39            42.279 ftdi_sio.ko
05/08/2011  10:39               698 liberror.c
05/08/2011  10:39               270 liberror.h
05/08/2011  10:39             8.774 librobot.c
05/08/2011  10:39             6.474 librobot.h
05/08/2011  10:39             3.700 libserie.c
05/08/2011  10:39               371 libserie.h
05/08/2011  10:39             1.183 Makefile
05/08/2011  10:39               497 stddef.h
05/08/2011  10:39             2.607 trapezoidal.c
05/08/2011  10:39             2.153 velocity.c
              13 archivos         72.438 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\findcolor

05/08/2011  10:39             3.459 findcolor.c
05/08/2011  10:39               148 Makefile
               2 archivos          3.607 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot

05/08/2011  10:39    <DIR>          client
05/08/2011  10:39    <DIR>          experimentos
05/08/2011  10:39    <DIR>          server
               0 archivos              0 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client

05/08/2011  10:39    <DIR>          c
05/08/2011  10:39    <DIR>          glade
05/08/2011  10:39    <DIR>          java
05/08/2011  10:39    <DIR>          mono
05/08/2011  10:39    <DIR>          php
               0 archivos              0 bytes
```

```
 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client\c

05/08/2011  10:39             1.689 client.c
05/08/2011  10:39             1.667 clientasyn.c
05/08/2011  10:39               563 Makefile
05/08/2011  10:39                18 resultado
               4 archivos          3.937 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client\glade

/08/2011  10:39    <DIR>          robotglade
               0 archivos              0 bytes

 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client\glade\robotglade

05/08/2011  10:39             1.067 AssemblyInfo.cs
05/08/2011  10:39               825 gui.glade
05/08/2011  10:39               692 Main.cs
05/08/2011  10:39               120 make.sh
05/08/2011  10:39               921 Makefile.robotglade
05/08/2011  10:39               567 robotglade.cmbx
05/08/2011  10:39               117 robotglade.mdsx
05/08/2011  10:39             2.333 robotglade.pidb
05/08/2011  10:39             2.996 robotglade.prjx
               9 archivos          9.638 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client\java

05/08/2011  10:39             1.349 client.java
05/08/2011  10:39           108.476 xmlrpc-1.2-b1.jar
               2 archivos        109.825 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client\mono

05/08/2011  10:39    <DIR>          canvasrobot
05/08/2011  10:39    <DIR>          simple
               0 archivos              0 bytes

 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client\mono\canvasrobot

05/08/2011  10:39    <DIR>          bin
05/08/2011  10:39               573 canvasrobot.cmbx
05/08/2011  10:39               117 canvasrobot.mdsx
05/08/2011  10:39             4.436 canvasrobot.pidb
05/08/2011  10:39             2.777 canvasrobot.prjx
05/08/2011  10:39            98.304 CookComputing.XmlRpc.dll
05/08/2011  10:39               121 make.sh
05/08/2011  10:39             1.092 Makefile.canvasrobot
05/08/2011  10:39               650 Proxy.cs
05/08/2011  10:39               554 Robot.cs
05/08/2011  10:39             1.647 Ventana.cs
05/08/2011  10:39            11.853 ventana.glade
05/08/2011  10:39            11.853 ventana.glade.bak
05/08/2011  10:39               261 ventana.gladep
05/08/2011  10:39               261 ventana.gladep.bak
              14 archivos        134.499 bytes

 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client\mono\canvasrobot\bin

05/08/2011  10:39    <DIR>          Debug
               0 archivos              0 bytes
```

```
 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client\mono\canvasrobot\bin\Debug

05/08/2011  10:39            20.480 canvasrobot.exe
05/08/2011  10:39            98.304 CookComputing.XmlRpc.dll
               2 archivos        118.784 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client\mono\simple

05/08/2011  10:39             1.078 App.ico
05/08/2011  10:39             2.426 AssemblyInfo.cs
05/08/2011  10:39    <DIR>          bin
05/08/2011  10:39             1.068 Class1.cs
05/08/2011  10:39             4.474 cliente.csproj
05/08/2011  10:39             1.822 cliente.csproj.user
05/08/2011  10:39               899 cliente.sln
05/08/2011  10:39             8.192 cliente.suo
05/08/2011  10:39            98.304 CookComputing.XmlRpc.dll
05/08/2011  10:39    <DIR>          obj
               8 archivos        118.263 bytes

 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client\mono\simple\bin

05/08/2011  10:39    <DIR>          Debug
               0 archivos              0 bytes

 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client\mono\simple\bin\Debug

05/08/2011  10:39            16.384 cliente.exe
05/08/2011  10:39            13.824 cliente.pdb
05/08/2011  10:39            98.304 CookComputing.XmlRpc.dll
               3 archivos        128.512 bytes

 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client\mono\simple\obj

05/08/2011  10:39    <DIR>          Debug
               0 archivos              0 bytes

 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client\mono\simple\obj\Debug

05/08/2011  10:39            16.384 cliente.exe
05/08/2011  10:39            13.824 cliente.pdb
05/08/2011  10:39             3.464 cliente.projdata
               3 archivos         33.672 bytes


 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\client\php

05/08/2011  10:39               477 cliente.php
05/08/2011  10:39            28.420 IXR_Library.inc.php
               2 archivos         28.897 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\experimentos

05/08/2011  10:39    <DIR>          E1
05/08/2011  10:39    <DIR>          E2
               0 archivos              0 bytes
```

```
 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\experimentos\E1

05/08/2011  10:39             6.169 apachec
05/08/2011  10:39             2.841 apacheclargo
05/08/2011  10:39             6.195 apachejava
05/08/2011  10:39             6.207 apachemono
05/08/2011  10:39             6.139 apachephp
05/08/2011  10:39             1.119 a_c
05/08/2011  10:39               527 a_cl
05/08/2011  10:39             1.145 a_j
05/08/2011  10:39             1.157 a_m
05/08/2011  10:39             1.089 a_p
05/08/2011  10:39           791.552 Excel.sxc
05/08/2011  10:39               307 get.pl
05/08/2011  10:39            34.782 resultados.sxc
05/08/2011  10:39             6.224 standalonec
05/08/2011  10:39             6.178 standalonejava
05/08/2011  10:39             6.177 standalonejava2
05/08/2011  10:39             6.160 standalonemono
05/08/2011  10:39             6.140 standalonephp
05/08/2011  10:39             1.159 s_c
05/08/2011  10:39             1.128 s_j
05/08/2011  10:39             1.127 s_j2
05/08/2011  10:39             1.110 s_m
05/08/2011  10:39             1.090 s_p
              23 archivos        895.722 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\experimentos\E2

05/08/2011  10:39               517 cliente2.php
05/08/2011  10:39            28.420 IXR_Library.inc.php
05/08/2011  10:39               390 salida_epia_apache_local
05/08/2011  10:39               391 salida_epia_apache_remoto
05/08/2011  10:39               391 salida_epia_apache_remoto_trucada
05/08/2011  10:39               400 salida_epia_boa_local
05/08/2011  10:39               403 salida_epia_boa_remoto
05/08/2011  10:39               382 salida_gimli_apache_local
05/08/2011  10:39               385 salida_gimli_apache_remoto
05/08/2011  10:39               384 salida_gimli_boa_local
05/08/2011  10:39               384 salida_gimli_boa_local_trucada
05/08/2011  10:39               386 salida_gimli_boa_remoto
              12 archivos         32.833 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\server

05/08/2011  10:39    <DIR>          cgi
05/08/2011  10:39    <DIR>          standalone
               0 archivos              0 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\server\cgi

05/08/2011  10:39               698 liberror.c
05/08/2011  10:39               270 liberror.h
05/08/2011  10:39             8.774 librobot.c
05/08/2011  10:39             6.474 librobot.h
05/08/2011  10:39             3.700 libserie.c
05/08/2011  10:39               371 libserie.h
05/08/2011  10:39               612 Makefile
05/08/2011  10:39             1.536 server.c
05/08/2011  10:39               497 stddef.h
05/08/2011  10:39             2.153 velocity.c
              10 archivos         25.085 bytes
```

```
 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\server\standalone

05/08/2011  10:39    <DIR>          conf
05/08/2011  10:39               698 liberror.c
05/08/2011  10:39               270 liberror.h
05/08/2011  10:39             8.774 librobot.c
05/08/2011  10:39             6.474 librobot.h
05/08/2011  10:39             3.700 libserie.c
05/08/2011  10:39               371 libserie.h
05/08/2011  10:39               617 Makefile
05/08/2011  10:39            79.100 server
05/08/2011  10:39             2.380 server.c
05/08/2011  10:39               497 stddef.h
              10 archivos        102.881 bytes

 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\server\standalone\conf

05/08/2011  10:39    <DIR>          abyss_root
               0 archivos              0 bytes

 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\server\standalone\conf\abyss_root

05/08/2011  10:39    <DIR>          conf
05/08/2011  10:39    <DIR>          htdocs
05/08/2011  10:39    <DIR>          log
               0 archivos              0 bytes

 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\server\standalone\conf\abyss_root\c
onf

05/08/2011  10:39             1.822 abyss.conf
05/08/2011  10:39             7.354 mime.types
               2 archivos          9.176 bytes

 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\server\standalone\conf\abyss_root\h
tdocs

05/08/2011  10:39               677 index.htm
05/08/2011  10:39             2.198 pwrabyss.gif
               2 archivos          2.875 bytes

 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gdrbot\server\standalone\conf\abyss_root\l
og

05/08/2011  10:39                 5 abyss.pid
05/08/2011  10:39         8.984.823 access.log
               2 archivos      8.984.828 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gpbot

05/08/2011  10:39    <DIR>          pc_side
05/08/2011  10:39    <DIR>          programas_asm
05/08/2011  10:39    <DIR>          programas_c
               0 archivos              0 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gpbot\pc_side

05/08/2011  10:39    <DIR>          ClientePC_GPBOT
               0 archivos              0 bytes
```

```
 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gpbot\pc_side\ClientePC_GPBOT

05/08/2011  10:39             1.740 clientepc.c
05/08/2011  10:39             7.048 gdrbot.c
05/08/2011  10:39             2.240 gdrbot.h
05/08/2011  10:39               304 Makefile
               4 archivos         11.332 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gpbot\programas_asm

05/08/2011  10:39               533 gpmap.inc
05/08/2011  10:39             2.105 gpregs.inc
05/08/2011  10:39                87 link08
05/08/2011  10:39               839 Makefile
05/08/2011  10:39             1.655 modelo.asm
05/08/2011  10:39             1.492 portb-sal.asm
               6 archivos          6.711 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gpbot\programas_c

05/08/2011  10:39    <DIR>          libs
05/08/2011  10:39    <DIR>          progs
05/08/2011  10:39    <DIR>          tests
               0 archivos             0 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gpbot\programas_c\libs

05/08/2011  10:39               661 libadc.c
05/08/2011  10:39               132 libadc.h
05/08/2011  10:39             2.089 libsci.c
05/08/2011  10:39             1.233 libsci.h
05/08/2011  10:39             2.498 libsrf04.c
05/08/2011  10:39               445 libsrf04.h
05/08/2011  10:39            25.918 mc68hc908gp32.h
05/08/2011  10:39               221 stddef.h
               8 archivos         33.197 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gpbot\programas_c\progs

05/08/2011  10:39    <DIR>          SSmonitor
               0 archivos             0 bytes

 Directorio de
D:\Datos\Varios\GdR\gdrbot\HEAD\src\gpbot\programas_c\progs\SSmonitor

05/08/2011  10:39               661 libadc.c
05/08/2011  10:39               132 libadc.h
05/08/2011  10:39             2.089 libsci.c
05/08/2011  10:39             1.233 libsci.h
05/08/2011  10:39             2.498 libsrf04.c
05/08/2011  10:39               445 libsrf04.h
05/08/2011  10:39               553 Makefile
05/08/2011  10:39            25.918 mc68hc908gp32.h
05/08/2011  10:39             1.070 SSmonitor.c
05/08/2011  10:39            18.370 SSmonitor_old.c
05/08/2011  10:39               221 stddef.h
              11 archivos         53.190 bytes

 Directorio de D:\Datos\Varios\GdR\gdrbot\HEAD\src\gpbot\programas_c\tests

05/08/2011  10:39             1.768 adc.c
05/08/2011  10:39                 0 ejemplo
05/08/2011  10:39               535 ejemplo.c
05/08/2011  10:39               824 Makefile
05/08/2011  10:39            25.701 mc68hc908gp32.h
05/08/2011  10:39             2.086 sci.c
05/08/2011  10:39             1.233 sci.h
```

```
05/08/2011  10:39            18.372 servidor-gpbot.c
05/08/2011  10:39             2.173 timer.c
               9 archivos         52.692 bytes

     Total de archivos en la lista:
             163 archivos     11.593.367 bytes
             128 dirs  27.748.605.952 bytes libres
```

Ficheros generados en el PFC, para los Servidores Sensoriales.

```
 Directorio de D:\Datos\Docencia\PFCs\Ismail\Doc definitiva\Contenido del CD

18/01/2010  18:39    <DIR>          API para el maestro
18/01/2010  18:39    <DIR>          Aplicación de ejemplo
18/01/2010  18:39    <DIR>          Código del módulo de comunicaciones A
18/01/2010  18:39    <DIR>          Código del módulo de comunicaciones B
18/01/2010  18:39    <DIR>          Código del módulo de locomoción
18/01/2010  18:39    <DIR>          Código del módulo US&IR
18/01/2010  18:39    <DIR>          Librerías para control de comunicaciones B
desde el PC
               0 archivos

 Directorio de D:\Datos\Docencia\PFCs\Ismail\Doc definitiva\Contenido del
CD\API para el maestro

24/11/2009  19:48             2.848 comunicaciones_A.c
24/11/2009  19:37             3.575 comunicaciones_A.h
23/11/2009  16:11             7.441 comunicaciones_B.c
24/11/2009  19:54            11.141 comunicaciones_B.h
24/11/2009  20:45             5.427 decodificador.c
23/11/2009  15:16             4.064 locomocion.c
24/11/2009  19:15             8.323 locomocion.h
27/11/2009  00:36             4.895 USI_Master.c
27/11/2009  00:32             5.378 USI_Master.h
23/11/2009  15:25             2.414 US_IR.c
24/11/2009  18:58             4.514 US_IR.h
              11 archivos         60.020 bytes

 Directorio de D:\Datos\Docencia\PFCs\Ismail\Doc definitiva\Contenido del
CD\Aplicación de ejemplo

05/12/2009  16:46            24.630 Dibujo.bmp
07/12/2009  20:06            13.274 firmware.c
27/11/2009  03:16             2.578 Makefile
               3 archivos         40.482 bytes

 Directorio de D:\Datos\Docencia\PFCs\Ismail\Doc definitiva\Contenido del
CD\Código del módulo de comunicaciones A

23/11/2009  03:53             1.649 comunicaciones.c
23/11/2009  03:53             3.492 comunicaciones.h
23/11/2009  03:53             1.850 firmware.c
12/11/2009  22:04             2.263 Makefile
23/11/2009  04:02             7.698 USI_Slave.c
23/11/2009  04:02             8.121 USI_Slave.h
               6 archivos         25.073 bytes

 Directorio de D:\Datos\Docencia\PFCs\Ismail\Doc definitiva\Contenido del
CD\Código del módulo de comunicaciones B

23/11/2009  04:13             1.904 comunicaciones.c
23/11/2009  04:13             3.450 comunicaciones.h
07/08/2009  18:47            10.669 delay_x.h
05/12/2009  17:01             6.987 firmware.c
23/11/2009  04:33             1.641 imagen_logos.h
23/11/2009  04:14             8.487 lcd.c
```

```
23/11/2009  04:36            12.200 lcd.h
23/11/2009  04:39             3.262 letras.h
11/11/2009  15:23             2.311 Makefile
23/11/2009  04:59             4.933 TWI_Slave.c
23/11/2009  04:59             6.027 TWI_Slave.h
              11 archivos         61.871 bytes

 Directorio de D:\Datos\Docencia\PFCs\Ismail\Doc definitiva\Contenido del
CD\Código del módulo de locomoción

22/11/2009  21:05             2.521 firmware.c
22/11/2009  21:52             5.563 locomocion.c
22/11/2009  21:43            10.784 locomocion.h
22/11/2009  20:48             2.250 Makefile
22/11/2009  20:56             7.699 USI_Slave.c
22/11/2009  20:56             8.100 USI_Slave.h
               6 archivos         36.917 bytes

 Directorio de D:\Datos\Docencia\PFCs\Ismail\Doc definitiva\Contenido del
CD\Código del módulo US&IR

22/11/2009  22:33             2.312 firmware.c
23/11/2009  03:46               957 generador_tabla.c
22/11/2009  22:44             3.127 infrarrojos.c
22/11/2009  22:40             4.362 infrarrojos.h
08/11/2009  01:53             2.464 Makefile
26/11/2009  01:20             4.407 tablas_magicas.h
22/11/2009  22:49             6.097 ultrasonidos.c
22/11/2009  22:54             6.253 ultrasonidos.h
23/11/2009  03:45             7.696 USI_Slave.c
23/11/2009  03:45             8.095 USI_Slave.h
              10 archivos         45.770 bytes

 Directorio de D:\Datos\Docencia\PFCs\Ismail\Doc definitiva\Contenido del
CD\Librerías para control de comunicaciones B desde el PC

05/12/2009  17:46             6.738 LCD_IRB.h
17/09/2009  02:35             5.370 LCD_linux_IRB.a
17/09/2009  01:37             5.074 LCD_win_IRB.lib
05/12/2009  17:26             1.006 Makefile
               4 archivos         18.188 bytes

     Total de archivos en la lista:
              65 archivos     12.992.918 bytes
              41 dirs  27.748.519.936 bytes libres
```

Anexo V. Piezas para la plataforma basada en GumStix

A.V.

Para el montaje de la estructura mecánica de la plataforma descrita en el punto 4.3 del capítulo 4, basada en Gumstix, se diseñaron y fabricaron una serie de piezas, ya presentadas en dicho capítulo. A continuación se muestra el detalle de cotas de cada una de ellas.

Todas las piezas, excepto las especificadas, están fabricadas en aluminio de 3 mm de espesor.

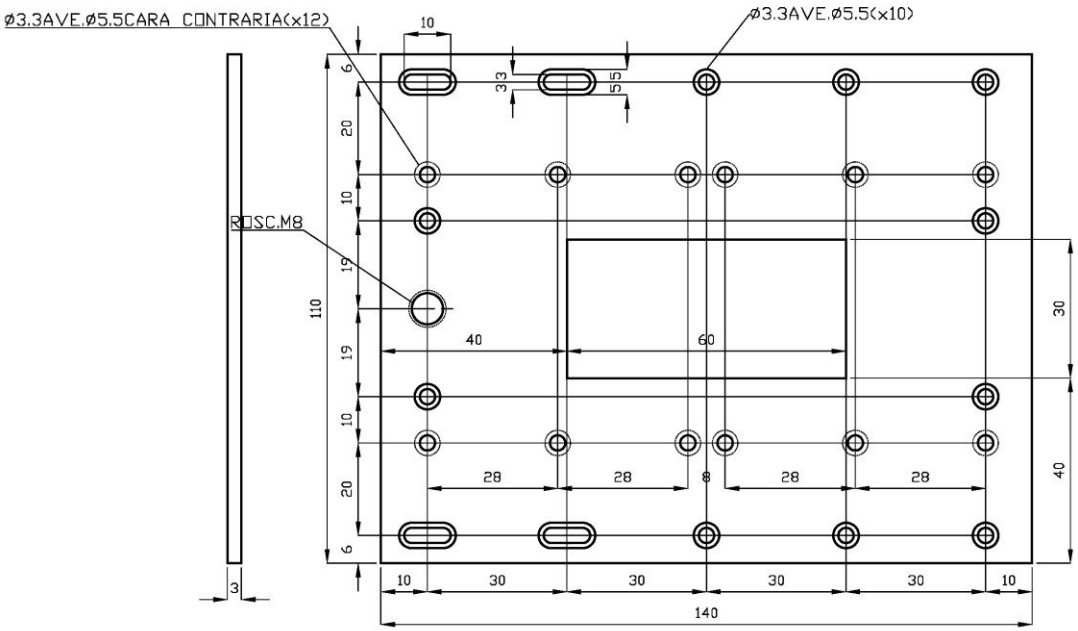


Figura AV.1. Plancha base de montaje

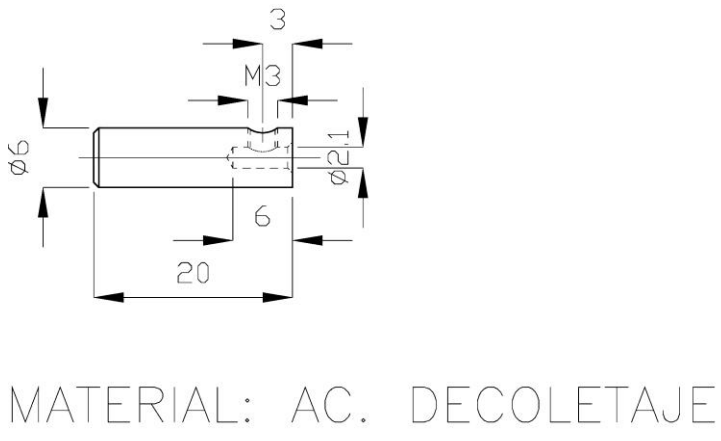


Figura AV.2. Adaptador de ejes motor-polea

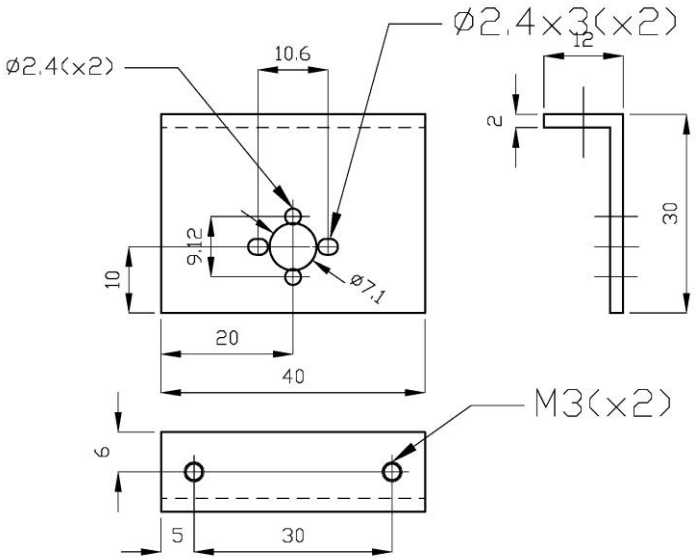
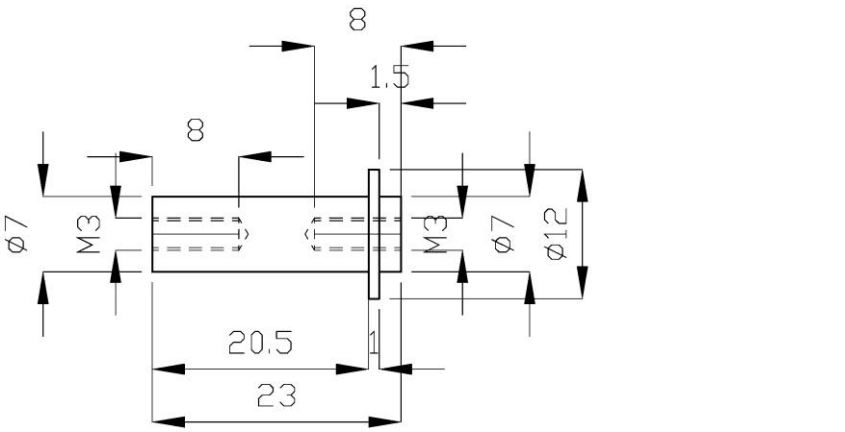


Figura AV.3. Soporte para motor



MATERIAL: AC. DECOLETAJE

Figura AV.4. Adatador de ejes rueda libre

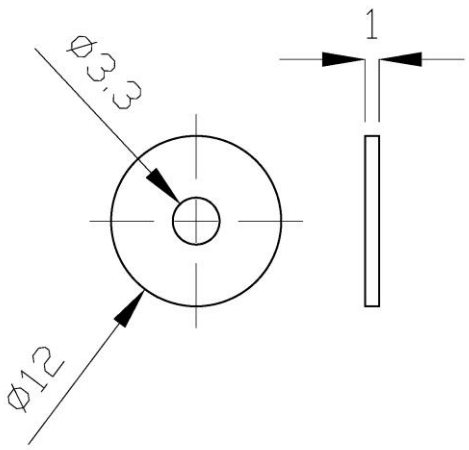


Figura AV.5. Arandela para la rueda libre

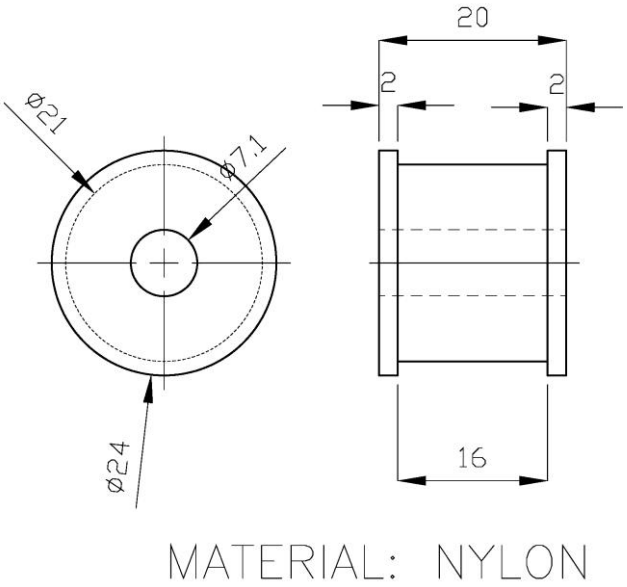


Figura AV.6. Polea para adaptar la correa

Presidente: Dr. Eduardo Boemo Scalvinoni

Secretario: Dr. Eloy Anguiano Rey

Vocal: Dr. Antonio García Ríos

Vocal: Dr. Jesús Bobadilla Sancho

Vocal: Dr. Guillermo Botella Juan